March 1980 Third Edition 50p

LIVERPOOL SOFTWARE GAZETTE

PILOT takes off...
LSG

During the Autumn of 1979 a small group consisting of successful publishers and acclaimed computer experts sat down to plan a new publication. Their brief? To produce the best all-round computer magazine on the market. Three months later the publication was born. It's name?

Computer Age

Computer Age is still only four issues old but already it is acknowledged by thousands to have achieved its original aim. Each new issue is enjoying increased sales and advertising revenue – but that's only part of the success story. More and more respected professionals are adding their names to the list of contributors and, one supposes, to an ongoing success story.

Whatever your interest in computers, the Publishers of Computer Age are confident you will be satisfied with the publication.

Why not put them to the test and see a sample copy? Just send 60p (to include postage and packing), the issue you would like to see (listed below) and full details to: COMPUTER AGE, 4 VALENTINE PLACE, LONDON S.E.1. A small price to pay for keeping in touch with the computer age!

Computer Age
The Electronic Office

ISSUE 1

Clive Jenkins and the unions' views. Are games playing programs intelligent? Artificial intelligence and the layman. Micro computers in education. Algorithmic designs. How mighty micros can aid super sales. Why not a computer in your business? Some microcomputer uses in psychology. Learning FORTH. Office of the future, and much more.

Computer Age
Artificial Intelligence
Computer Aided Learning
Languages
Accountants' Guidelines
Solicitors' requirements
First time business users

ISSUE 2

The government's views on computers. Cultural energy and the personal computer. Tribute to Christopher Evans. Microcomputers in the legal professions. Accountants' guide to microcomputers. The peripheral connection. Topics in artificial intelligence. Why not a computer in your school? Brand news, and much more.

Computer Age
Faceless bureaucrats — an invasion of privacy?
Computer Assisted Learning
Computers for first timers
Accountants' guidelines
Actual users

ISSUE 3

Faceless bureaucrats – an invasion of privacy? Buying a personal computer. Statistical analysis on a micro. Doing it in binary. The small systems market hots up. Book reviews. When we're all ten years older. Database. The modular approach. Optimising employment.

Computer Age
Christopher Bodder-Williams

ISSUE 4

The microcomputer and the teacher. Computer retailing. Using a digital ring. Desert island floppy discs. Computers for the smaller firm. Systems for small businesses. Keeping your computer busy. New technologies for education. More on FORTH.

> **"This periodical can be highly recommended on the evidence of its first two issues. It contains for the businessman, professional firm, teacher and individual a reliable guide almost equivalent to a computer Which on the ins-and-outs of buying and using computers."**
>
> *The Times Monday February 18*

CONTENTS

Page 4 EDITOR AND PUBLISHERS LETTER
The Editor airs his views, not too contentious this month we hope.

Page 6 PETS CORNER
John Stout continues his excellent regular series applicable to 20% of all micros in use in the U.K.

Page 16 NASCOM NOTES
Michael Shanahan airs his views and helps Nas-sys users to test their memory.

Page 21 CARTOON

Page 22 APPLE PIPS
Our regular series on Apple is a little smaller this month but articles elsewhere more than make up for this.

Page 24 AIM 65 ASSEMBLER
Find out more about this vastly under-rated piece of kit and 6502 programming in general.

Page 31 GRAPHICS SHAPE SCALING
Our contributing Editor will have more to say on this next month

Page 32 PILOT TAKES OFF
A full listing of a Pilot Assembler in BASIC

Page 42 LETTER FROM AMERICA
The first of Dave Smith's regular features on the latest West Coast computer gossip.

Page 44 PASCAL—AN INTRODUCTION
Dr. Andrew Veronis starts his lucid explanation on this structured high level language.

Page 48 PROGRAMMING PRACTICES AND TECHNIQUES
This regular series continues this month with a look at editors.

Page 51 CARTOON

Page 52 ALGO 68C ON THE Z80
In this article Raymond Anderson seeks to point out that Pascal is not the only structured high level language that will run on a micro.

Page 58 MICROCOMPUTERS AND BIO-CHEMISTRY
This application article helps to show that microcomputers are tools for a wide variety of possible useages.

Page 62 SHARP MACHINE LANGUAGE
An introduction to using the Z80 within your Sharp

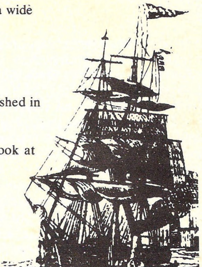
Page 65 ERRATA

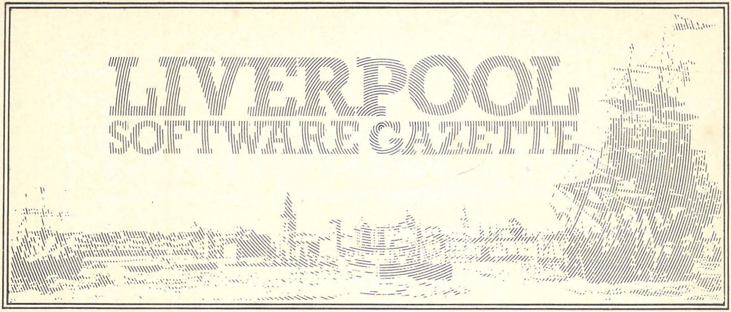
Page 66 SUPER SORT
Roy Stringer thoroughly dissects the workings of his sort programme published in issue 2 of this magazine.

Page 70 SOFTWARE AND ACORN
The only regular feature for Acorn owners continues this month with a look at interfacing.

Page 74 NUMERICAL ACCURACY
Dr. Allan continues from last month on this essential topic.

Page 78 ETC.
It looks like hardware is infiltrating into the Gazette.





Editor

0.0

WHAT an amazing industry this is, a few short years ago computers were the preserve of 'professionals', looked upon by outsiders in awe and deference. The advent of the micro has liberated the computer, almost overnight computer time has gone from being a precious commodity to being freely and cheaply available to anyone who wants. The result has been a broad base of computer literacy and nothing demonstrates this better than the fact that the authors of articles in this magazine include an architect, a biochemist and even a salesman. We welcome this democratisation of computing as it is resulting in an almost infinite variety of ideas and applications that can only be good for the industry and thus for society as a whole. Certainly we look forward to publishing many more articles from non-computer professionals.

0.1

THIS issue of the magazine sees the introduction of two more regular features, both by American authors. Letter from America is written by Dave Smith who has been in micro-computers from the beginning, in Southern California and who has his finger on the pulse of the Orange County computer scene. His monthly contribution will do much to keep out readers aware of the way the industry is going. Andrew Vernois is a computer professional who has worked for several of the American microelectronic giants and who has written quite a few books on computing/microelectronics. We hope that bringing his professionalism to spreading the Pascal gospel will help more people realise that BASIC is not the ultimate way to use their machine.

With successive issues we hope to carry more regular features. The next issue should see a 6800 section and a regular home for news and information on the Sharp MZ 8OK, we hope too to find sufficient material for a regular 1802 feature as we feel this excellent processor has not received the coverage it deserves.

0.2

A high level language is a tool for doing a job and just as a hammer is unsuitable for putting in a screw, so different languages are suitable for different jobs and a programmer needs a tool box of different languages to apply to the different tasks. In this issue are articles about two of the tools that are available, Algol, as well as our regular series on Pascal. In future issues there will be further articles on these and other languages, whilst obviously continuing to support BASIC as this is the microcomputer language in the most widespread use.

Further up and coming goodies are an application article about microcomputers in architecture, a look at Crystals superb Z80 BASIC, an analysis of the Social Effects of Microcomputers and articles on the various Sorcerer Pacs, among many others. To maintain and even enhance the quality of this journal we need reader feedback, even if just to complain, so don't hesitate in taking pen to paper.

Publisher

1.0

LIVERPOOL Software Gazette is a company in its own right, separate from Microdigital, though it is necessary for us to continue using Microdigital's resources for a while. This should make no difference, our editorial content has not favoured Microdigital in any way as we have always regarded the Gazette being a separate entity. We hope this change will encourage more advertising in the Gazette, because advertising revenue is the lifeblood of a magazine. As a further inducement to advertise we are giving every advertiser (even ¼ page) 50 free copies of the issue they advertise in, selling these will help recoup part of the advertising cost.

1.1

IF you are at an academic institution or a member of a computer club or any other group that can get together a regular bulk order for the Gazette you will be able to buy at 'trade' price from:-

Computer Bookshop,
43-45 Temple Street,
Birmingham.

1.2

First Edition	52 pages
Second Edition	68 pages
This Edition	84 pages
Next Edition	100 pages???

This magazine has more pages of articles than any other microcomputer magazine published in the U.K.

B. Everiss

BRUCE EVERISS

SHARP Computer

A personal computer that opens the world of programming to your own fresh ideas!

SHARP MZ-80K

CRT Display
This unit is equipped with a 25 cm (10") monochrome CRT for up to 1.000 letters (40 letters × 25 lines) Processing results can be displayed on the CRT. and it is possible to program and edit (addition, deletion, etc.) while watching the operation for confirmation

A Technical Masterpiece
A personal computer that makes full use of the multi-functions of an 8-bit microcomputer (Z-80), this model is certainly one of the most advanced anywhere It employs BASIC language, a feature which provides easy programming even to those totally unfamiliar with computer operation.

78 Keys
ASCII standard
Alphabet (capital and small letters)
Graphic symbols

Built-In Clock
Clock circuit time is displayed according to program

SHARP MZ80K SOFTWARE

BREAKOUT	£5.00	SUPER SIMON	£5.00
MASTERMIND	£5.00	MIZ-MAZE	£8.00
SHAPE MATCH	£5.00	GRAPHICS/MUSIC	£5.00
LUNAR LANDER	£5.00	BOMBER	£5.00
SNAKES & LADDERS	£5.00	FIREBALLS	£5.00
		DONKEY DERBY	£5.00

HB COMPUTERS LTD

22 NEWLAND STREET, KETTERING, NORTHANTS.
Tel. (0536) 83922 & 520910 Telex 341297

LIVERPOOL SOFTWARE GAZETTE

Editor/Publisher: Bruce Everiss.
Editorial Assistant: Nikki Devereux.
Contributing Editors: John Shout, Dr Martin Beer, Dave Straker.
Advertising/Subscriptions: Nikki Devereux.
Artwork: Peter Croft, John Burrows.

ADVERTISING:

Full page (17.5 cm x 24 cm)	£180.00
Half page (Upright 8.5 cm x 24 cm)	£ 95.00
Half page (Landscape 17.5 cm x 11.75 cm)	£ 95.00
Quarter page (8.5 cm x 11.75 cm)	£ 52.00
Agency discount 10%	

Please contact Nikki Devereux on 051-227 2535 if you would like further details.

DISCLAIMER: 'All the information in the magazine has been throughly debugged and tested. However, no guarantees are made as to its truth or validity'.

TRADE DISTRIBUTION:— Computer Bookshop, 43-45 Temple Street, Birmingham. 021-643 4577.

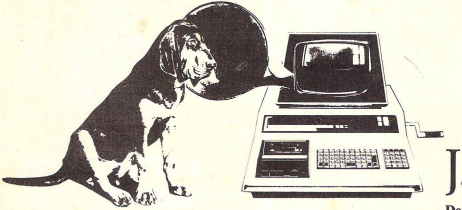
U.S. DISTRIBUTION:— Bits Inc., P.O. Box 428, 25 Route 101 West, Peterborough, NH 03458.

THE LIVERPOOL SOFTWARE GAZETTE is published bi monthly by Liverpool Software Gazette Limited, 14 Castle Street, Liverpool L2 0TA. Registered in England No 1477285

Subscriptions: Within Great Britain, £6.00 for 12 issues. Individual copies, by post 60p. Please tell us the issue you would like to start a subscription with!

REPRINTS: Articles that are explicitly marked as having restricted reproduction rights may not be copied or reprinted without written permission from Microdigital. All other articles may be reprinted for any non-commercial purpose provided a credit line is included stating that said material was reprinted from the Liverpool Software Gazette, 14 Castle Street, Liverpool, L2 0TA. Please send copies of any reprints to Liverpool Software Gazette, attention of Bruce Everiss.

Pets Corner



J.Stout

Department of Architecture
Liverpool Polytechnic
53 Victoria Street
Liverpool
L1 6EY

IN the last issue a routine was given which, when passed the first byte of 6502 machine code instruction in the accumulator, returned the number of bytes in that instruction (1..3). It did not check the instruction for being a valid instruction (103 out of the 256 possible instructions are invalid instructions) and so could not be used for an application such as a single step program.

The short routine below, using 50 bytes, returns with the carry bit set if the byte passed to it in the accumulator represents a valid machine code instruction, otherwise it returns with the carry bit clear. It uses 32 bytes of data, where the i'th byte of data (0 < = i < = 31) has a 1 in bit j (0 < = j < = 7, bit 0 being the least significant bit) if the machine code instruction represented by (8* i + j) is a valid one. In the next issue a relocator using both of these routines will be listed, and details given for linking it into the Commodore provided machine code monitors, either in ROM or from tape.

```
VALID: 48       PHA           ;save op-code
       29 07    AND  #$07     ;mask for low order
                               bits. (A) now equiv-
                               alent to j above,
                               giving bit number
                               within byte i

       AA       TAX
       E8       INX           ;(X)=j+1
       68       PLA           ;get op-code again
       4A       LSR  A
       4A       LSR  A
       4A       LSR  A        ;bits 7..5 now zero,
                               last 3 instructions
                              ;divide op-code by 8
       A8       TAY           ;(Y) now equivalent
                               to j above
       B9 4C 03 LDA  TABLE,Y  ;the only instruction
                               in the routine which
                              ;is NOT relocatable
SHIFT: 4A       LSR  A        ;if valid/invalid bit
                               was in bit 0 it is now
                              ;in carry? which is
                               where we want it
       CA       DEX
       D0 FC    BNE  SHIFT    ;if (X) non-zero then
                               need to shift again
       60       RTS           ;done so return

TABLE: 63 67 63
       63 73 77
       63 63 63
       77 63 63
       63 77 63
       63 72 75
       73 27 77
       77 73 77
       73 77 63
       63 73 77
       63 63
```

The routine above is assembled to occupy the first 50 bytes of the second cassette buffer (033A ..03F9), which gives a value of 034C for the address TABLE. If the routine is to be moved elsewhere the instruction LDA TABLE,Y will of course need altering, since we have not at the moment written a relocator program to do it for us.

Searching

IN the last issue the topics of searching and sorting were brought up, and an expansion promised for this issue. Rather than try and deal with both topics at once we will deal with the searching problem first, and start with a description of the problem we are attempting to solve.

A variable KEY is given, together with an array FILE, which contains NE number of elements. The task is to find that index IN such that FILE (IN) = KEY. The definition of FILE will have occurred in a statement such as DIM FILE (NE) at the start of the problem, where NE may either have been assigned a value as the result of an INPUT statement or of an assignment statement. Wherever the routines presented refer to the size of the array they will use NE, so that the problem does not depend on a particular value of NE (not that users of PETs with old ROMs must have NE less than or equal to 255).

A typical application might be to find the location in the array which contains the number KEY, then use that position to retrieve other information linked to that number, e.g. a description of a part with that given part number, where that extra information may be stored in another array, e.g. DES$(NE).

In the special case where the range of possible values for KEY is 1..NE then of course our task is much simplified, since all we would need to do is to look straightaway at DES$(KEY). The problem of searching simple 'squashes' KEY to fit into the range of possible values for an array. At the end of this article we shall present an application which takes a command letter, or group of letters, and generates an index IN which can be used as the subject of an ON GO TO (GOSUB).

In the case when KEY is not an element of FILE (), then we return a value of IN = 0. This means that we cannot use the zero'th element of FILE, but should it be necessary the algorithms presented can be adapted to return -1, or some other distinguishable value.

First effort (Listing 1)

The slightly unusual construct in line 1020 makes sure that the FOR loop is properly terminated. In some cases it may not be needed (and may well not work for other implementations of BASIC or in other languages) but it is always better to be safe than sorry. If the program terminates in line 1040 we know that FILE (I)< >KEY, I = 1 . .NE, and so IN = 0 (or -1, or whatever value is chosen to represent failure). If FILE (I) = KEY, then IN is set to that value of I, the FOR loop terminated and the routine ended.

Second effort (Listing 2)

To cut down on the size of the routine we might notice that we have two occurences of NEXT I:RETURN. We cannot simply delete the first occurrence from line 1020, since the routine would then always return IN = 0. However, if we move the IN = 0 to *before* the start of the FOR loop, then the algorithm works and is slightly shorter.

This method of assigning a value to a variable before a test which if true assigns another value to it can be used to replace some IF . . .THEN . . .ELSE . . .statements, encountered when moving programs from one implementation of BASIC to another. Thus IF (X = 0) THEN A = B ELSE A = C can be written as A = C: IF (X = 0) THEN and for true compatibility every statement before the IF must have a corresponding one in the THEN section.

Note: a) we could remove the I = NE statement from line 1020, but this has two effects:

i) if KEY occurs more than once in FILE, IN returns the highest value of I such that FILE(I) = KEY

ii) the loop is not terminated once the value KEY is found, so that the routine takes longer to execute when KEY is found.

b) it is unlikely that we have speeded the routine up by much, since the alterations have only been to statements that are only executed once. If the subroutine were repeatedly called it might be worthwhile, but all we have done so far is to perhaps reduce the size of the routine and perhaps make it slightly more efficient.

Third effort (Listing 3)

If one is unable to use a FOR loop to control the number of iterations of the main part of the subroutine, but must implement one using IFs and GOTOs, perhaps when searching through a linked list (another source for at least one article), then a straight 'translation' of the second effort might be Listing 3i.

However notice that two IF statements will be executed each time round the loop. One way to reduce this is to make use of the zero'th element in FILE as follows in Listing 3ii.

At some time the condition in line 1020 *must* be false, since if the element KEY was not in FILE(NE) . .FILE (1) it is definitely in FILE(0) by reason of the second statement in line 1010. If KEY was in FILE(NE) . .FILE(1) then the loop will exit with IN equal to the relevant value, which is what we want.

This technique of using a particular element to act as a 'sentinel' for the loop, is equivalent to using a variable with an abnormal value to signal the end of a list of data. It's basic use is to simplify the terminating condition of the routine, or as we have seen, to remove one or more of the terminating conditions altogether. It has a certain elegance in that we generate the value of IN = 0 automatically, without having to assign it especially.

Ofter a long sequence of IF statements occurs in a program, where a variable is being tested against a large number of possibilities, and the one which is truly being used to either GOTO a different portion of the program or to assign different values to other variables. The linear search algorithms introduced above can be used to simplify this type of programming quite easily. Perhaps the best way to demonstrate it is to consider the action of a combined text editor and formatter program, which enables the user to input text to the program, interspersed with formatting commands, e.g. to set the left margin, centre a line, set the number of characters per line and so on. One convention used in Kernighan and Plaugher's Software Tools (Addison Wesley) is that a formatting directive begins with a .and is the first thing on a line. They are two letter commands, e.g. .bp, .br, .ce, .ul and so on. The first task is to determine that a formatting command is to be obeyed, and then to get the two letter string which then compared with the possible range of values for the formatting command, these possibilities having been loaded into an array earlier on in the program. One of the search algorithms above is then used to see if the command obtained is one of the legal commands. If it is then we return IN = a value in the range 1 . . NC, where NC is the number of commands. This value of IN can then be used in an ON IN GOTO (GOSUB) command.

There are a number of slight variations on this technique. When using strings the target strings may be concatenated into one long string and the command string compared against substrings of this string. This normally needs some sentinel string at the end, e.g. XX, or the idea of tacking the command string onto the end, as in the routine in Listing 3ii.

What we are doing is to create a form of CASE statement, equivalent to an ON GOTO (GOSUB) where the argument is not numerical. The search is being used to map the possible range of values of the command string onto the integers 1 . .NC. Listing 4 demonstrates the technique for Kernighan and Plaugher's 14 formatting commands, using the one long string technique.

So far we have concentrated on altering the routines in ways which are unlikely to confer much benefit on the timing of the routines. On average, assuming the values to be evenly distributed in FILE, we will do NE/2 comparisons before finding KEY (if it is in the array), otherwise we will do NE (+ 1 in the sentinel version). Thus the effort of the routine is proportional to the number of elements in the file, double the number of elements and you double the number of comparisons. Given that the array is sorted we can do much better than this with a technique known as the binary search.

Binary Search (Listing 5)

This method works by approximately halving the range of the array to be looked at for each comparison, and looking at the middle element (or the element nearest the middle). Since we assume the array is ordered (such that i less than j implies FILE (i) less than or equal to FILE (j)), there are three possibilities for the result of the comparison. Firstly the comparison could yield true, in which case we have found the element we were looking for. Secondly the middle element is less than KEY. In this case we know that if KEY is in the array it must be in the top half of the array, and we can repeat the technique on the top half of the array. Thirdly the middle element is greater than KEY in which case we repeat the technique on the lower half of the array. The important thing to note is that at each iteration we halve the range under consideration, so that the number of comparisons will be of the order of $\log^2(NE)$. Thus double the number of elements and we add a constant number of comparisons, the number necessary to determine whether we have found the element, must search the lower half of the range or the upper half.

As we have said, this technique depends on the array being sorted, but for large arrays its performance is so much better than the linear search that it will often repay the extra effort made. It shows that to speed a process up gains will not be made from removing spaces, comments etc which can even approximate those made by altering the algorithm. If a program is not fast enough one's first question should be 'Is there a better algorithm?'

There exists an even faster method of searching (though it is not useful in every situation) which depends not on the data being sorted but on it exhibiting a peculiar form of randomness. The technique of hashing will be covered in the next issue along with the problem of sorting (the Supersort presented in the last issue uses a form of hashing to achieve its spectacular performance, along with a large amount of memory).

A USEFUL RANDOM NUMBER GENERATOR

Sometimes it can be useful to generate random numbers in such a way that one is sure of getting all the possible values, e.g. in a game. For real numbers this is impossible, but in the situation where the numbers in the range $0 \,..\, 2^m-1$ are to be generated the random number generator in listing 6 can be useful (and we will return to it in next issue's discussion of hashing).

The generator works as follows:

1) start with initial 'seed' R = 5.
2) multiply R by 5 and place the result in R.
3) mask out all but the lowest m + 2 bits of R (equivalent to taking the remainder when dividing by 2^{m+2}).
4) the next random number is INT(R/4).
5) repeat from step 2.

The routine in listing 6 generates 6, 7, 4, 5, 2, 3, 0 and 1, and then repeats.

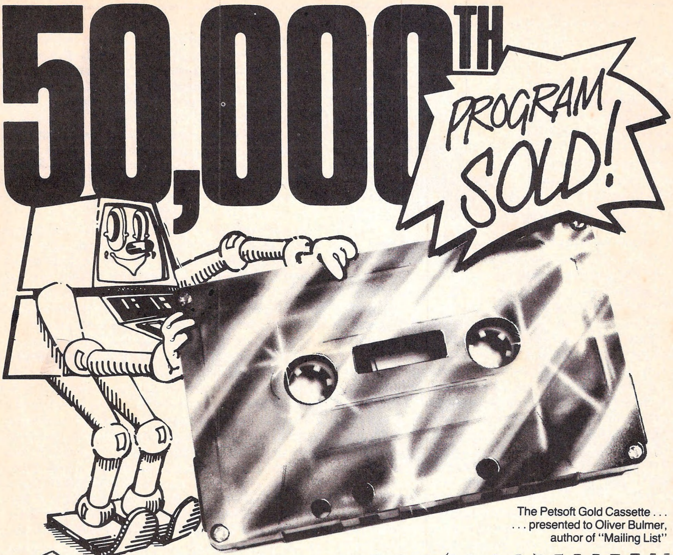
Recursion

Recursion is the process of defining something in terms of itself. The usual and overworked example is that of the factorial function, which can be defined as:

factorial (n) = n*factorial (n-1), and factorial (0) = 1.

Since we can calculate factorial (n) using iteration (F = 1: FOR I = 2 TO N:F = F*N:NEXT I), we could do with a better example of the power of recursion. Consider the game THE TOWER OF HANOI (or The Trilogic Game to old Doctor Who fans). Given 3 poles with no disks of ascending size (biggest at the bottom) slipped over pole 1, move them to pole 3, moving only 1 at a time, and at no time having a larger disk on top of a smaller (pole 2 can be used as an intermediate one). A formula for the number of moves is not hard to find, 2 disks—3 moves, 3 disks—7 moves, 4 disks—15 moves, but a BASIC program not using recursion is rather difficult to write and is rather difficult to follow. Consider instead the recursive procedure written in Pascal:

```
procedure move (numberofdisks,from,on-
            to,via:integer);
begin if numberofdisks  1 then writeln ('Move disk
      from', from, 'to',onto)
else begin move (numberofdisks-1,from,via,onto);
           writeln ('Move disk from',from,'to', onto);
```



The Petsoft Gold Cassette . . .
. . . presented to Oliver Bulmer,
author of "Mailing List"

60% OFF!

We celebrated by slashing Ledger systems prices by over **60%**:

SALES LEDGER ______ **£95**

PURCHASE LEDGER ______ **£95**

Developed by ACT, Britain's leading computing group, to run on a 32K PET with Anadex or Datac BD80 printers. Commodore Disk versions available price £115.

These systems provide full facilities for ledger maintenance, preparation of lists of outstanding balances, printing of statements and remittance advices. Full audit trail. Send for details.

PLUS

Disk Payroll £50 for up to 200 employees
Disk Stock Control £50 handling 2,500 stock items (Petsoft/CompuThink Disk) or 400 stock items (Commodore Disk)

AND

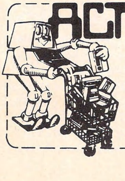
Mailing List £15
Word Processor £25
PET BASIC Tutorial £15
Assembler/Editor £25
VAT Pack £17.50
Invoicing £20
Forth £30
Statistics £7
Microchess £14
Super Startrek £8
Eliza Doctor £8
Backgammon 8

Prices exclude VAT. Credit card orders accepted by telephone. All programs available through your local PET dealer or direct from:

All prices correct at the time of going to Press.
PET is the trademark of Commodore

ACTPetsoft

Radclyffe House, 66-68 Hagley Road, Edgbaston, Birmingham
B16 8PF. Telephone: 021-455 8585 Telex: 339396



Please rush me your latest catalogue of over 170 PET programs.

My name is . . . .

I live at . . . .

. . . . Postcode . . . .

I have a new/old ROM PET I have NO PET

```
          move (numberofdisks-1,via,onto,from)
                                    end
End
```

What the procedure says is that to move 5 disks from pole 1 to pole 3 we firstly move 4 disks to pole 2 (via), move the remaining disk on 1 (from) to 3 (onto), then move the 4 disks on 2 (via) to 3 (onto) using 1 (from) as the intermediary. Thus the procedure reduces a problem to a slightly easier version of itself. Eventually the task will be to move 1 disk, in which case the procedure can handle that itself without calling on itself with a slightly easier problem.

All recursion must exhibit these two main features, a call to a simpler version of itself and a section which does not include a call to itself. If the second part is missing the recursive process will never end. In our example the number of disks being moved is 1 less each time a recursive call is made, so that eventually (assuming numberofdisks to be positive) a call will be made to move which does not result in a recursive call.

There is nothing magical about recursion, and it can be performed in BASIC, but we need to explicitly write some of the housekeeping routines which are performed automatically in systems supporting recursion. The main task is to make sure that the recursive call does not destroy any values which will be needed upon return from the call, and the natural mechanism for doing this is a stack, which we simulate in BASIC with an array (of reals, integers and/or strings, depending on what type of variables we need to save) and a pointer to that array. This pointer is incremented by 1 (or as many spaces as we need) upon entry to the routine and the save variables put into the space allocated. Upon exit from the subroutine the pointer is decremented and the saved variables (or rather their values) restored.

The selection of subroutines (listing 7) exhibit indirect recursion, where routine A calls B which calls A. Their purpose is to convert an expression written in standard algebraic notation into Reverse Polish notation (familiar to users of HP, Novus and the early Scientific calculators). In R. P. notation A + B becomes AB +, (A + B)*(C-D) becomes AB + CD-* and A + B*C becomes ABC* + (assuming normal precedence). The main use for R. P. is that the evaluation of an expression is very easy, and is performed in the following way:

1) start at the left most character of the expression, set SP = 0
2) get the next character. If it is the end-of-line marker exit
3) if it is a variable put the value associated with it into the array RE at element SP and increment SP by 1
4) if it is an operator perform that operation on the top operands in array RE (i.e. RE(SP-2)operatorRE (SP-1),)put the result in RE (SP-2) and decrement SP by 1
5) move to the next character in the expression and goto step 2

upon exit the result of the expression will be in RE(0). If SP doesn't equal 1 then there was an error in the original expression.

Compilers (and probably interpreters) convert the algebraic form of an expression into R. P. (interpreters implicitly generate the result at the same time as converting it), and the routines are presented as an aid to writing a calculator package, or perhaps adding calculator facilities to a PILOT interpreter.

We describe the syntax of the expressions the routines will convert by using syntax diagrams, which are mostly commonly used to describe the syntax of Pascal, but which will display very neatly the relationship between the syntax of the expressions and the routines which convert to R. P.

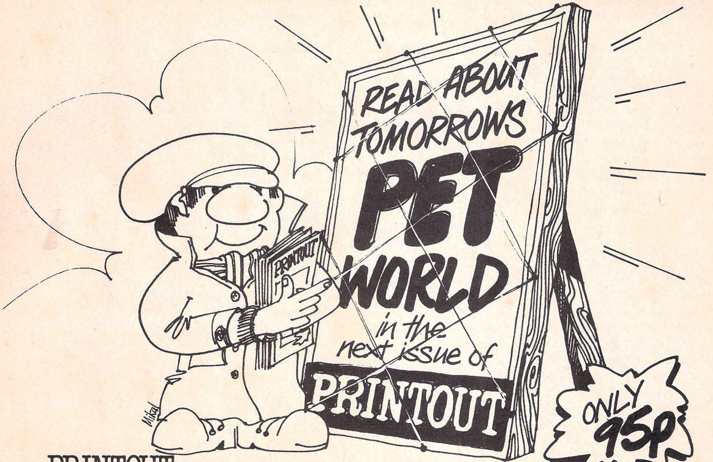
At the moment we will only make the routines convert simple expressions, that is only expressions involving single letter variables (A . .Z) and the 4 operators *, /, + and -, together with (and) for altering precedence, into R. P.. The general principle is that on entry to one of the routines we have in CH$ the first character of the string which it is to translate. Thus is the input is (A = B) on calling the routine at 2000 CH$ = ")". From the syntax diagrams the first component of an expression is a term, so we call subroutine 3000.

Since a term consists of a factor the first job of 3000 is to call 4000 which then checks for the presence of a "(". If there is one then factor knows that it is being called upon to evaluate an expression, which it does by calling the expression subroutine, 4000, after first finding the first character of the expression by calling 1000. The structure of the routines models very closely that of the syntax diagrams, and it will be noticed that the only places where a stack is needed to hold the value of a variable is where there are two or more possible entry points to a recursive call, e.g. in the expression subroutine, where if the first term has been dealt with and either + or - has been met, in which case whichever it was must be remembered so that on return the relevant operator may be added to the end of the R. P. expression. Since every element to be dealt with is a single character we are stacking them by concatenating the character on to the end of a string and on return removing the character from the string.

One possible problem with recursion, especially with the PET and all 6502 based machines is that BASIC uses the stack internal to the 6502 to store return addresses, and since the 6502 stack is limited to 256 bytes this places a limit on the number of recursive calls that can be made. A simple program:

```
10 N = 0
20 N = N + 1: GOSUB 20
```

reveals that when the out of memory error occurs N = 26. If the out of memory error occurs when a PRINT FRE (0) reveals many bytes free then this is usually the problem, GOSUBs using up the entire stack space available. FOR loops also use up stack space so in certain programs less than 26 levels of subroutines can cause the error. For a particular application the only real way to find out whether an application can work is to try it. The Tiny Pascal compiler mentioned in the first issue runs



PRINTOUT, THE PET USERS MAGAZINE, hits the streets 10 times a year bringing you the very latest news, reviews and pictures of PET products and software, plus programming tips and listings in a glossy professionally produced format.

Already PRINTOUT's worldwide network of correspondents have broken exclusive stories on the next generation of PET computers and peripherals currently under development.

In the past four issues PRINTOUT:

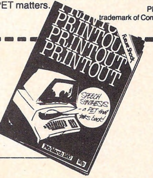
- ★ Consumer tested and reported on over one hundred games programs.
- ★ Looked at a low cost speech synthesis system and reported on Commodore's upcoming voice synthesiser.
- ★ Evaluated the pros and cons of both Commodore and CompuThink disk systems.
- ★ Carried an exclusive report on Commodore's new colour PET.
- ★ Printed detailed stories on PET applications in Education, Model Railways and Public Relations.
- ★ Test rated programming aids and utilities.
- ★ Evaluated Word Processors for the PET.
- ★ Published a full length article on Modular Programming and complete listing of the MPAK program.
- ★ Examined the workings of PET's keyboard and video logic.
- ★ Printed excerpts from two authoritative new books–*The Hitchhikers Guide to the PET* and *PET Revealed* 2nd edition.
- ★ Published complete program listing and documentation for high density plotting on the PET.
- ★ Surveyed Business software packages.
- ★ Reported on input routines for the PET.

COMING SOON:

- ★ Which Printer? Consumer report
- ★ String Handling
- ★ The Ultimate Assembler
- ★ Hard disk systems–The Pros and Cons
- ★ Micro Networking–The Good News
- ★ Are you ready for Stringy Floppy?
- ★ Human Engineering for your programs

PRINTOUT, a single source for what you need to know about PET computing. Because we are independent, we can bring you the news first, honestly and without bias. Already PRINTOUT is quoted and reprinted in the other magazines as *the* authority on PET matters.

PET is the trademark of Commodore



SUBSCRIBE TO: PRINTOUT

PRINTOUT, Greenacre House, North Street, Theale, Berkshire RG7 5EX
Please enter my subscription for one year (10 issues)

I enclose: ☐ £9.50 U.K. ☐ £10.50 Eire ☐ £14.50 Overseas ☐ 95p Sample Issue

My name is ______

My address is ______

______ Postcode ______

PET Configuration ______

without encountering this problem so most applications should have no trouble.

NOTES: There is no error checking, but this could be added, e.g. testing that CH$ = ")" in line 4010 after GOSUB 2000. If the input routine 1000 were replaced by calls to the scanner of the last issue then multiple character variables could be coped with, together with : = ,<>, > = etc. The basic idea for these routines comes from the Pascal User Manual and Report, page 73 . .75.

Timeout in INPUT

One possible use of the line input routine listed in last issue, lines 50100 to 50220, is to return to the calling routine with a variable set if the user does not reply within a certain time. This could be especially useful for computer aided instruction packages where if the user of the package does not reply to an answer within a certain time the program is able to recognise this and perhaps provide a prompt or hint. Should all the hints have been provided the program could then give the answer and an explanation of how this was arrived at. The length of time allowed could be made a function of the hardness of the problem. Even on simple business programs the timing out could be considered to indicate that the user does not know how to respond to the question and results in an explanation of what input is required.

The changes are only small, and are summarised below:

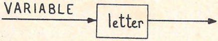
```
50110 L$= " " : TS=TI : TT=0 : REM LINE IS INITIALLY
        NULL, TS IS TIME ON ENTRY, TT=0 INDICATES
        NO TIMEOUT.
50120 POKE 167,0: GET KY$: IF (KY$<>" ") THEN
        50130
50122 IF (TI-TS)> TC) THEN TT=1:RETURN: REM TT=-1
        INDICATES TIMEOUT.
50124 GOTO 50120:REM TC IS THE  LENGTH OF TIME
        ALLOWED FOR RESPONSE  IN JIFFIES.
```

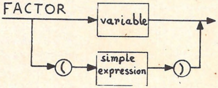
If on return TT =-1 then a timeout has occurred (note that this can be tested by simply IF TT THEN . . .). If the timeout is to be ignored if the user has typed anything at all then the condition in 50122 should become ((TI-TS) > TC)AND (L$ = " ").

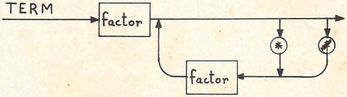
I can be reached at the following address:

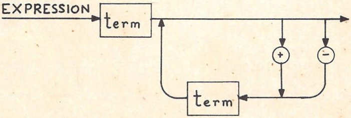
6 College Ave., Formby, Merseyside. L37 3JJ.

For telephoning during working hours please try 051 236 0598.









LISTING 1

```
 1000 REM SEARCH FOR KEY IN FILE(NE).
 1010 FOR I=1 TO NE
 1020 IF (FILE(I)=KEY) THEN IN=I:I=NE:NEXT I:GOTO 1040:REM FOUND.
 1030 NEXT I:IN=0:REM NOT FOUND.
 1040 RETURN
READY.
```

LISTING 2

```
 1000 REM SEARCH FOR KEY IN FILE(NE).
 1010 IN=0:FOR I=1 TO NE
 1020 IF (FILE(I)=KEY) THEN IN=I:I=NE
 1030 NEXT I:RETURN:REM IN=0=>NOT FOUND.
READY.
```

LISTING 3i

```
 1000 REM SEARCH FOR KEY IN FILE(NE).
 1010 IN=0:I=1
 1020 IF (FILE(I)=KEY) THEN IN=I:GOTO 1040
 1030 I=I+1:IF (I<=NE) THEN 1020
 1040 RETURN
READY.
```

LISTING 3ii

```
 1000 REM SEARCH FOR KEY IN FILE(NE).
 1010 IN=NE:FILE(0)=KEY
 1020 IF (FILE(IN)<>KEY) THEN IN=IN-1:GOTO 1020
 1030 RETURN
READY.
```

LISTING 4

```
 10 NC=14:FC$="":FOR I=1 TO NC:READ CD$:FC$=FC$+CD$:NEXT I
 20 DATA BP,BR,CE,FI,FO,HE,IN,LS,NF,PL,RM,SP,TI,UL:REM SET UP COMMAND STRING.
 30 INPUT "¬OMMAND▮▮▮.▮▮▮▮";KEY$
 40 GOSUB 1000:IF (IN=0) THEN PRINT "¯RROR: COMMAND NOT RECOGNISED.":PRINT:GOTO
30
 50 PRINT "¬OMMAND NUMBER";IN:PRINT:GOTO 30
 1000 REM SEARCH FOR 2 LETTER COMMAND KEY$ IN FC$. SINCE COMMANDS ORDERED COULD
 1010 REM USE BINARY SEARCH, SEE LISTING 5.
 1020 IN=0
 1030 FOR CN=1 TO NC
 1040 IF (MID$(FC$,CN*2-1,2)=KEY$) THEN IN=CN:CN=NC
 1050 NEXT CN:RETURN
READY.
```

LISTING 5

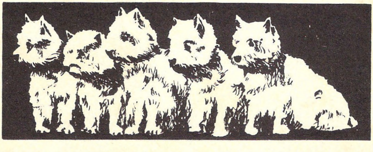
```
 1000 REM BINARY SEARCH FOR KEY IN FILE(NE), ASSUMES SORTED.
 1010 IN=0:LEFT=1:RIGHT=NE:REM FIRST SECTION TO SEARCH IS ENTIRE ARRAY.
 1020 IF (LEFT>RIGHT) THEN 1070:REM NOT FOUND.
 1030 MIDDLE=INT((LEFT+RIGHT)/2):TRY=FILE(MIDDLE)
 1040 IF (KEY=TRY) THEN IN=MIDDLE:GOTO 1070
 1050 IF (KEY<TRY) THEN RIGHT=MIDDLE-1:GOTO 1020
 1060 LEFT=MIDDLE+1:GOTO 1020
 1070 RETURN
READY.
```

LISTING 6

```
 10 R=5:M=3:REM SET SEED AND RANGE OF NUMBERS WANTED.
 20 FOR I=0 TO (2↑M)-1
 30 GOSUB 1000:PRINT RN,
 40 NEXT I:PRINT:PRINT:GOTO 20
 1000 REM GENERATE NEXT RANDOM NUMBER IN RN.
 1010 R=R*5:REM STEP 2.
 1020 R=(R)AND(2↑(M+2)-1):REM STEP 3.
 1030 RN=INT(R/4):REM STEP 4.
 1040 RETURN
READY.
```

LISTING 7

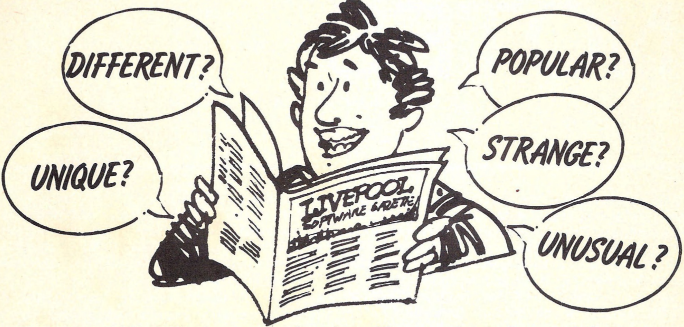
```
 10 POKE 59468,14:PRINT "⊐⊢ONVERSION TO _EVERSE ⌝OLISH.":PRINT:PRINT
 20 GOSUB 5000:REM READ IN EXPRESSION.
 30 GOSUB 1000:REM FIND FIRST NON-SPACE CHARACTER.
 40 OP$="":SE$=" ":TE$=" ":GOSUB 2000:REM INITIALISE RESULT, STACKS AND GET AN
 50 REM EXPRESSION.
 60 PRINT OP$:IF (CH$<>".") THEN 20
 70 END
 1000 REM GET NEXT NON-SPACE, NON-END OF LINE CHARACTER FROM IP$.
 1010 CP=CP+1:IF (CP>L) THEN GOSUB 5000:GOTO 1010
 1020 CH$=MID$(IP$,CP,1):IF (CH$=" ") THEN 1010
 1030 RETURN:REM WITH CH$.


 2000 REM SIMPLE EXPRESSION.
 2010 GOSUB 3000:REM SORT OUT A TERM.
 2020 IF (CH$<>"+")AND(CH$<>"-") THEN 2050
 2030 SE$=SE$+CH$:GOSUB 1000:GOSUB 3000:OP$=OP$+RIGHT$(SE$,1):SE$=LEFT$(SE$,LEN(
SE$)-1)


 2040 GOTO 2020
 2050 RETURN:REM SE$ USED AS STACK FOR OPERATOR BETWEEN 2 TERMS AT THIS LEVEL.
 3000 REM TERM.
 3010 GOSUB 4000:REM GET A FACTOR.
 3020 IF (CH$<>"*")AND(CH$<>"/") THEN 3050
 3030 TE$=TE$+CH$:GOSUB 1000:GOSUB 4000:OP$=OP$+RIGHT$(TE$,1):TE$=LEFT$(TE$,LEN(
TE$)-1)


 3040 GOTO 3020
 3050 RETURN
 4000 REM FACTOR.
 4010 IF (CH$="(") THEN GOSUB 1000:GOSUB 2000:GOTO 4030:REM RECURSION.
 4020 OP$=OP$+CH$:REM NOT ( SO ASSUME VARIABLE.
 4030 GOSUB 1000:RETURN:REM COULD ECONOMISE ON STACK USAGE HERE BY GOTO 1000.
 5000 INPUT "⌝XPRESSION■■■.■■■";IP$:IP$=IP$+"↑":CP=0:L=LEN(IP$):RETURN
 5010 REM CONCATENATING IP$ WITH "↑" IS NEEDED SO PROGRAM ENDS. TRY IT WITHOUT.
READY.
```



Reach the people who matter with the

LIVERPOOL SOFTWARE GAZETTE

The microcomputer magazine that is


DIFFERENT?
POPULAR?
UNIQUE?
STRANGE?
UNUSUAL?
LIVERPOOL SOFTWARE GAZETTE

An informal poll of our readership (3 samples) showed that they were well educated, normal, responsible people who buy things.

It therefore obviously pays to advertise . . .

Ridiculously reasonable rates high quality editorial and production standards make us unbeatable media in which to advertise **your** product, however mythical it may be. Advertise that Z-8000 board here, the £325 1200 CPS impact Printer, or Hard Disk sub-system for SC/MP.

Notes

Full page	(17.5 cm x 24 cm)	£180.00
Half Page	(Upright 8.5 cm x 24 cm)	£95.00
	(Landscape 17.5 cm x 11.75 cm)	£95.00
Quarter Page	(8.5 cm x 11.75 cm)	£52.00

Agency discount 10%

Copy Date for the May issue is April 18
Copy Date for the July issue is June 20

ALL ADVERTISERS RECEIVE 50 COPIES OF THE ISSUE THEY ADVERTISE IN, FREE OF CHARGE

14 Castle Street
Liverpool L2 OTA
Tel: 051-227 2535/6/7/8

NASCOM NOTES

AT the present moment Nascom are still supplying Nascom 2's with free 16K R.A.M. Kits. This has caused some problems. Firstly, as mentioned in Nascom Notes last time, the recommended position for the four EPROM'S in the Memory Map was F000 to FFFF. This was to fit in with Tiny Basic and Super Tiny Basic.

But on a Nascom 2 the 8K Basic lies from E000H to FFFFH. There is a one line note somewhere in the Nascom 2 documentation telling you to relocate the EPROMS at D000H to DFFF but I'm still getting Nascom's back because due to this error they won't run 8K Basic (obviously from fools who don't buy the Gazette)!

The second problem is that the two memory test programs in the back of the memory board construction notes were written to run under Nasbug T2 or T4. They call three monitor subroutines as part of the error routine. The addresses of these must be changed to run under Nas-Sys 1. Re-assembled versions of the two programs are shown below.

N.B. If you have a perfect RAM Board (some chance!) and have run these programs you probably won't have had any problems, since the wrong addresses only occur in the error routine, which is never called if there are no RAM faults!

Incidentally the programs were listed on one of the New Nascome Imp Printers (our specimen was a bit drunk at the time!) Talking of which, if you try to run an IMP off a Nascom 1 you will run into problems by just following the steps in the manual (good ol' Nascom). It's all very well feeding the external UART clock from the printer into the Nascom 1's external clock pin, but it wont go anywhere unless the on board UART clock select link is changed to external! Since it needs to be changed back to internal for reading cassettes it might be as well to fit a SPDT switch instead of a link. (Try writing to Nascom for one under Warranty)

```
0010 ;
0020 ;  **************************************
0030 ;  *     MEMORY TEST PROGRAM FOR NASCOM.
0040 ;  *     THIS IS THE SAME PROGRAM AS
0050 ;  *     IN THE BACK OF THE CONSTRUCTION
0060 ;  *     NOTES , BUT WITH SUBROUTINES
0070 ;  *     CHANGED TO WORK WITH NAS-SYS 1.
0080 ;  *
0090 ;  *   PROGRAM WRITTEN USING ZEAP AND
0100 ;  *   PRINTED USING NASCOM IMP PRINTER
0110 ;  *
0120 ;  *      ZEAP IS £30.00 PLUS VAT
0130 ;  *      IMP IS £325.00 PLUS VAT
0140 ;  *
```

```
                 0150 ; *      MICRODIGITAL LTD
                 0160 ; *      25 BRUNSWICK STREET
                 0170 ; *      LIVERPOOL L2 0PJ
                 0180 ; *
                 0190 ; *     MIKE SHANAHAN   28/2/80
                 0200 ; ***********************************

                 0210 ;
                 0220 ;           TEST PROGRAM ONE
                 0230 ;    TESTS IF EACH BYTE IS UNIQUELY
                 0240 ;           ADDRESSABLE
                 0250 ;
0319             0260 B2HEX   EQU   £0319
                 0261 ;
                 0262 ;     B2HEX PRINTS CONTENTS OF ACC.
                 0263 ;     AS TWO HEX DIGITS.
                 0264 ;
0306             0270 SPACE   EQU   £0306
                 0271 ;
                 0272 ;     SPACE PRINTS A SPACE
                 0273 ;
0311             0280 CRLF    EQU   £0311
                 0281 ;
                 0282 ;     CRLF PRINTS A CARR. RETURN /
                 0283 ;     LINE FEED
                 0284 ;

0D00             0290         ORG   £0D00
0D00 0600        0300 TEST    LD    B,00
0D02 3E4F        0310         LD    A,£4F
0D04 32E00B      0320         LD    (£0BE0),A
0D07 2A0E0C      0330 LOOP    LD    HL,(£0C0E)
0D0A ED5B100C    0340         LD    DE,(£0C10)
0D0E 13          0350         INC   DE
0D0F 7D          0360 PUTIN   LD    A,L
0D10 AC          0370         XOR   H
0D11 A8          0380         XOR   B
0D12 77          0390         LD    (HL),A
0D13 23          0400         INC   HL
0D14 B7          0410         OR    A
0D15 ED52        0420         SBC   HL,DE
0D17 19          0430         ADD   HL,DE
0D18 20F5        0440         JR    NZ,PUTIN
0D1A 2A0E0C      0450         LD    HL,(£0C0E)

0D1D 7D          0460 RDBACK  LD    A,L
0D1E AC          0470         XOR   H
0D1F A8          0480         XOR   B
0D20 4F          0490         LD    C,A
0D21 7E          0500         LD    A,(HL)
0D22 B9          0510         CP    C
0D23 C43C0D      0520         CALL  NZ,ERROR
0D26 23          0530         INC   HL
0D27 B7          0540         OR    A
0D28 ED52        0550         SBC   HL,DE
0D2A 19          0560         ADD   HL,DE
0D2B 20F0        0570         JR    NZ,RDBACK
```

```
0D2D 3AE00B   0580         LD   A,(£0BE0)
0D30 EE40     0590         XOR  £40
0D32 32E00B   0600         LD   (£0BE0),A
0D35 10D0     0610         DJNZ LOOP
0D37 EF       0620         RST  £28
0D38 2A       0630         DEFM /*/
0D39 00       0640         DEFB £00
0D3A 18C4     0650         JR   TEST
              0660 ;
              0670 ;    ERROR PRINTOUT
              0680 ; THE FORMAT IS :-
              0690 ;    ADDRESS  EXPECTED  FOUND
             0700 ;


0D3C E5       0710 ERROR   PUSH HL
0D3D D5       0720         PUSH DE
0D3E C5       0730         PUSH BC
0D3F F5       0740         PUSH AF
0D40 7C       0750         LD   A,H
0D41 CD1903   0760         CALL B2HEX
0D44 7D       0770         LD   A,L
0D45 CD1903   0780         CALL B2HEX
0D48 CD0603   0790         CALL SPACE
0D4B 79       0800         LD   A,C
0D4C CD1903   0810         CALL B2HEX
0D4F CD0603   0820         CALL SPACE
0D52 F1       0830         POP  AF
0D53 CD1903   0840         CALL B2HEX
0D56 CD1103   0850         CALL CRLF
0D59 010000   0860         LD   BC,00
0D5C 0B       0870 WAIT    DEC  BC
0D5D 78       0880         LD   A,B
0D5E B1       0890         OR   C
0D5F 20FB     0900         JR   NZ,WAIT
0D61 C1       0910         POP  BC
0D62 D1       0920         POP  DE
0D63 E1       0930         POP  HL
0D64 C9       0940         RET


              0010 ;
              0020 ; ************************************
              0030 ; *    MEMORY TEST PROGRAM FOR NASCOM.
              0040 ; *    THIS IS THE SAME PROGRAM AS
              0050 ; *    IN THE BACK OF THE CONSTRUCTION
              0060 ; *    NOTES , BUT WITH SUBROUTINES
              0070 ; *    CHANGED TO WORK WITH NAS-SYS 1.
              0080 ; *
              0090 ; *   PROGRAM WRITTEN USING ZEAP AND
              0100 ; *   PRINTED USING NASCOM IMP PRINTER
              0110 ; *
              0120 ; *     ZEAP IS £30.00 PLUS VAT
              0130 ; *     IMP IS £325.00 PLUS VAT
              0140 ; *
```

```
                   0150 ; *      MICRODIGITAL LTD
                   0160 ; *      25 BRUNSWICK STREET
                   0170 ; *      LIVERPOOL L2 0PJ
                   0180 ; *
                   0190 ; *     MIKE SHANAHAN   28/2/80
                   0200 ; ***********************************
                   0210 ;
                   0220 ;               TEST PROGRAM TWO
                   0230 ;    TESTS IF EACH BIT CAN BE SET
                   0240 ;                  AND RESET
                   0250 ;
0319               0260 B2HEX   EQU   £0319
                   0261 ;
                   0262 ;     B2HEX PRINTS CONTENTS OF ACC.
                   0263 ;     AS TWO HEX DIGITS.
                   0264 ;
0306               0270 SPACE   EQU   £0306
                   0271 ;
                   0272 ;     SPACE PRINTS A SPACE
                   0273 ;
0311               0280 CRLF    EQU   £0311
                   0281 ;
                   0282 ;     CRLF PRINTS A CARR. RETURN /
                   0283 ;     LINE FEED
                   0284 ;
0D00               0290         ORG   £0D00
0D00 0E00          0300 MTEST   LD    C,00
0D02 2A0E0C        0310 OUTER   LD    HL,(£0C0E)
0D05 ED5B100C      0320         LD    DE,(£0C10)
0D09 13            0330         INC   DE
0D0A 79            0340 INNER   LD    A,C
0D0B 77            0350         LD    (HL),A
0D0C 46            0360         LD    B,(HL)
0D0D B8            0370         CP    B
0D0E C4220D        0380         CALL  NZ,ERROR
0D11 23            0390         INC   HL
0D12 B7            0400         OR    A
0D13 ED52          0410         SBC   HL,DE
0D15 19            0420         ADD   HL,DE
0D16 20F2          0430         JR    NZ,INNER
0D18 0C            0440         INC   C
0D19 79            0450         LD    A,C
0D1A B7            0460         OR    A
0D1B 20E5          0470         JR    NZ,OUTER
0D1D EF            0480         RST   £28
0D1E 2A            0490         DEFM  /*/
0D1F 00            0500         DEFB  00
0D20 18E0          0510         JR    OUTER
                   0520 ;
                   0530 ;    ERROR PRINTOUT
                   0540 ; THE FORMAT IS :-
                   0550 ;    ADDRESS   EXPECTED   FOUND
```

```
                    0560 ;
0D22 E5             0570 ERROR   PUSH HL
0D23 D5             0580         PUSH DE
0D24 C5             0590         PUSH BC
0D25 7C             0600         LD   A,H
0D26 CD1903         0610         CALL B2HEX
0D29 7D             0620         LD   A,L
0D2A CD1903         0630         CALL B2HEX
0D2D CD0603         0640         CALL SPACE
0D30 79             0650         LD   A,C
0D31 CD1903         0660         CALL B2HEX
0D34 CD0603         0670         CALL SPACE
0D37 78             0680         LD   A,B
0D38 CD1903         0690         CALL B2HEX
0D3B CD1103         0700         CALL CRLF
0D3E 010000         0710         LD   BC,00
0D41 0B             0720 WAIT    DEC  BC
0D42 78             0730         LD   A,B
0D43 B1             0740         OR   C
0D44 20FB           0750         JR   NZ,WAIT
0D46 C1             0760         POP  BC
0D47 D1             0770         POP  DE
0D48 E1             0780         POP  HL
0D49 C9             0790         RET
```

CRYSTAL ELECTRONICS
CC ELECTRONICS

The newest Z80 Basic!

XTAL Basic 2·2

HAS to be the best yet for your Nascom 1or 2 All the usual features of other 8K floating-point BASICs **Plus: Extra commands/functions—INCH, KBD, CMD$ ON ERR GOTO, ERR, PI, CLOAD? (tape verify)**

And Add up to 64 reserved words of **your** choosing—**Now** put your own disc, tape, control, graphics commands, etc for the ULTIMATE in BASIC flexibility! Fully upward compatible with version 2.1 (see earlier Ads). Can be easily adapted to most Z80 systems. Works with T2, B-BUG, T4 **and** NAS-SYS monitors. Existing version 2.1 users—Return your original tape (less manual) with 50p P & P and we will update it FREE of charge!

Price: still only £35 + VAT!

CREED PRINTER INTERFACE

For NASCOM or APPLE—lowest cost hard copy! Complete kit of parts (with software) £18 + VAT.

16 CHANNEL RELAY BOARD

Now in stock for NASCOM 1/2. For £49.95 + VAT Sixteen switched (isolated) channels for many control applications. This kit will greatly increase the flexibility of your NASCOM.

Members of Computer Retailers Association & Apple Dealers Association

Shop open 0930-1730 except Wed. & Sun.
40 Magdalene Road, Torquay, Devon, England. Tel: 0803 22699

Access and Barclaycard welcome. COMPUTERS AND COMPONENTS

You've heard about it
Read about it — HERE IT IS

SCOOP OFFER NASCOM-2
ONLY 500 KITS AVAILABLE AT THIS PRICE

AVAILABLE EX-STOCK
COMPLETE KIT AS PER MANUFACTURER'S SPECIFICATION
With provision for 8K on board expansion. Excludes 4118 x8★

INCLUDES FREE 16K EXPANSION
VALUE £140 includes ALL parts with every kit

NASCOM-2
ON DEMONSTRATION NOW

£295 + 15% VAT **WITH FREE 16K EXPANSION WORTH £140**

AVAILABLE ONLY FROM US ON THE COUPON BELOW

OPTIONAL EXTRAS
3 AMP POWER SUPPLY VAT 15%
£29.50 Post £1.50
For NASCOM-2

8 OFF 4118★
For NASCOM-2
PLR Chasers
£80 Early Delivery

RS232 COMPATIBLE
80 COLUMN PRINTER brand new
List price £550. If sent by carrier £5 extra
OUR PRICE £325 + VAT 15%

FULL MANUFACTURER'S WARRANTY — DON'T DELAY, ORDER TODAY

Please send me my **NASCOM-2 KIT with the FREE 16K EXPANSION for £295 + VAT.**

I enclose remittance .... to cover

Name & Address ....

....

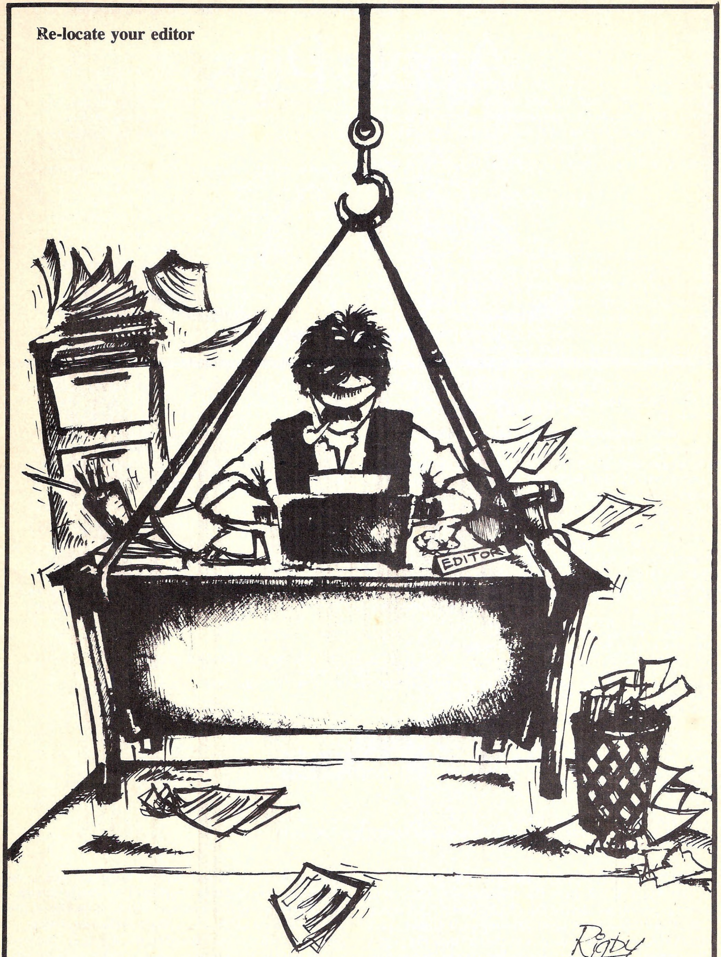
Also in stock NASCOM-1 ● ELF ● TRS80 as previously advertised

Buy with Access

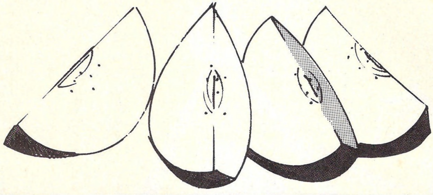
HENRY'S
Computer Kit Division
404 Edgware Road, London, W2, England
01-402 6822

BARCLAYCARD

Re-locate your editor


EDITOR
Rigby

Apple Pips



IN last issues Apple Pips we omitted to give the address of Owl Computers who are responsible for LISP on the Apple II.

For those who are interested in this package here is the address:—

Owl Computers,
41 Stortford Hall Park,
Bishops Stortford,
Hertfordshire,
CM23 5AJ.

For those of you who have been victims of the Apples reset key there is the heartening news that all new Apples now being imported into the U.K. have a separate 'Encoder' board directly underneath the keyboard.

This Encoder Board has a 2 position switch, in one position the reset functions normally and in the other position control reset is required to get any action.

```
?SYNTAX ERROR
]LIST
10  REM   ************************
20  REM   **  DATE VET PROGRAM  **
25  REM   **                    **
30  REM   **   BY G JONES       **
40  REM   ************************
50  DIM A(12): REM  A= MONTHS AS NUMBERS
60  DATA   31,28,31,30,31,30,31,31            ,30,31,30,31: REM  NO. OF DAYS PER MONTH
70  FOR I = 1 TO 12: READ A(I): NEXT I: REM  SET UP MONTH ARRAY
100  HOME : VTAB 10: INPUT "WHAT IS TODAY'S DATE(DD/MM/YY) ";X$: IF  LEN (X$) <  > 8 THEN 100
110  REM  INPUT DATE AS 8 CHARACTER STRING
115 A(2) = 28
120 DD% =  VAL ( LEFT$ (X$,2)): REM  DAY VALUE
130 MM% =  VAL ( MID$ (X$,4,2)): REM  MONTH VALUE
140 YY% =  VAL ( RIGHT$ (X$,2)): REM  YEAR VALUE
150  IF YY% < 80 THEN 100: REM  CHECK YEAR >= 80
160  IF MM% < 1 OR MM% > 12 THEN 100: REM  CHECK MONTH
170  FOR I = 0 TO 99 STEP 4
172  IF I = YY% THEN A(2) = 29
174  NEXT I
176  REM  ALLOW FOR LEAP YEAR
180  IF DD% < 1 OR DD% > A(MM%) THEN 100: REM  CHECK DAY FOR MONTH
200  VTAB 20: PRINT "DATE OK ": END
```

FROM APPLE MAGAZINE NUMBER 3

The Apple II will have the responsibility for one of many experiments aboard the 15 x 60 foot NASA Space Lab. It will monitor a plant-growing experiment, from which NASA hopes to piece together a scientific puzzle determining what effect gravity has on the mysterious 'Helical' spiraling patch followed by plant seedlings as they grow. Plant growth will be recorded with a videotape camera and other pertinent information, such as temperature and illumination, will be transmitted back to scientists on earth. Each phase of the experiment will be controlled and monitored by the Apple II. The overriding goal is to sharpen up on our experimental techniques in space.

UCLA Counts Sheep— For Mom's Sake

Researchers are hooking up expectant mothers to an Apple and keeping them there for 30 days at the UCLA 'University of California, Los Angeles) Medical School.

Sounds inhuman? It is. The mothers are sheep, not humans, and the purpose of the experiments, is to find out more about the role of the endocin system in the birth process.

The computer setup—which goes under the unwieldy name of Biophysical Variable Signal Processing System—assists in the monitoring of various processes in both the foetus and the mother, according to Dr. Kitch Wilson, a faculty member in the Department of Paediatrics and the father of the computer system. Transducers, he said, are implanted in the foetus and in the mother's uterus and blood veins to monitor everything from blood and inter-utherine pressure to oxygen and blood flow.

The sensors are attached to a black box that converts pressure readings into electrical signals and finally into digital numbers which are analysed by the computer and then stored on a disk and used later in correlatum analyses. Various experiments are performed before and during labour, while the sensors continue their monitoring function.

The final correlation analyses will be done by a larger minicomputer, but the Apple II does all the signal processing and the real-time acquisition and data analysis. It does it especially well. Wilson said, because it's interruptable.

'It's like having two computers. First, it does monitoring and housekeeping chores; then, every 10 milliseconds it's interrupted and goes down into the assembly language program where the input is processed. When that's finished it goes back to the BASIC program of monitoring.'

General Telephone of Pennsylvania—has come up with a solution to the problem of testing lines frequently enough at a reasonable cost. It uses an Apple II in a system developed by one of its employees, Ed Didion. Didion has programmed the Apple to test each of the company's 2000 trunks and produce a printout telling what the expected volume was on each, what the actual level was, the percentage of trunks that passed and failed the standard volume, and what the projected failure time is. This enables them to better plan their maintenance schedules.

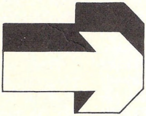
from The Apple Shoppe No 4.

APPLE KEYPAD

Advanced Business Technology has announced the Keypad; a 13 key keypad for the Apple computer. The keypad is in a molded plastic case matching the Apple case, and includes '.', '-', and 'return' keys in addition to the 10 key numerics. The pad interfaces with the normal Apple keypad in piggy-back fashion and costs $125.00. It is being distributed to Apple dealers in Southern California by the Apple distributor 0B-1, and should be available at your local Apple dealer.

APPLE AND THE VIDEO DISK

Utah State University has combined the Apple computer with the new video disk to create the ultimate teaching machine. Each side of the disk stores 54,000 screens of information. The information is retrieved using laser technology and any frame may be randomly accessed in 2.5 seconds. A full search of the disk takes only 5 seconds. A single side of the disk will thus hold the equivalent of 7.5 million bytes of storage. The December issue of Interface Age has an excellant article on this new technology, which eventually will revolutionize education if read/write capability can be economically produced.



ASSEMBLER PROGRAMMING ON THE AIM 65

Dr Martin D. Beer

Computer Laboratory
University of Liverpool
P.O. Box 147
Liverpool L69 3BX.

THE AIM-65 has been around for some time now. It was originally designed by Rockwell as a system design kit to allow engineers who were interested in using the 6502, which they were second-sourcing from MOS Technology to gain some experience in programming the microprocessor before designing their own systems. Although it is approximately twice the cost of similar design kits, it provides many additional facilities which make it suitable for use by the sophisticated amateur, as a home computer.

For a basic price of £265 (+ VAT) you obtain a main printed circuit board, containing the microprocessor, 1K of RAM, a 4K monitor in ROM a twenty character display, various 1/0 controllers from the 6502 range, and a little thermal printer. Input can be from a full ASCII keyboard supplied with the AIM-65, or from a 20 ma, current loop teletype connected to the peripheral port. As can be expected from a component manufacturer, a full set of 6502 manuals together with a complete circuit diagram, a very comprehensive User Manual and an Assembler Listing of the Monitor are also included. Sockets are provided to expand the RAM to 4K, and to install another three 4K x 8 ROMs, which can be used either for additional software provided by Rockwell, or, by the user's own PROMs.

Facilities are available for connecting up to two tape recorders for the storage of programs and data through the peripheral connector at the rear of the main board. Software is included in the monitor to load and store programs and data on the cassette tapes, and to turn the recorders on and off if the remote control leads are connected correctly. Should further memory, or 1/0 be required, it can be connected to the expansion connector, also at the rear of the computer. Rockwell have, very sensibly, made the peripheral and expansion connectors compatible with two other 6502 based single-board microcomputer systems. This means that there are already a wide range of expansion cards available which are, at least, hardware compatible with the AIM-65. In addition Rockwell have added a few interesting ones of their own, notably a Bubble Memory board.

As can be expected with a 4K monitor, a full ASCII keyboard, a built-in character display and a printer the programming facilities available on the basic AIM-65 are very extensive. Not only are the standard features of displaying and changing the contents of both registers and memory locations and the execution of user programs fully supported, but there are also a number of other more unusual facilities. Breakpoint can be set at desired instructions within the program. The breakpoint facility can be turned on and off without clearing the breakpoint settings.

This is a very useful facility since it is often necessary to run the program through to check for correct operation before finally clearing them. Separate instructions are included to trace instruction codes, the register contents, or the contents of the program counter. This is very useful since, usually, trace output is routed to the printer and the generation of unnecessary trace information would waste a lot of paper.

As an additional debugging aid a single-step facility is provided through a slide switch on the main computer board. Software is included to read and write to cassette tapes in both KIM 1 format, as used by the other single board computers, and the AIM-65's own format which gives a better transfer rate and is more reliable. Also within the monitor are a mnemonic instruction input mode and a disassembler.

These two instructions allow the user to input his programs directly as assembler codes, and avoid the necessity of hand-assembling programs. The disassembler can be used to check the contents of program memory and output is in the same form as the instructions were entered. Once developed user programs can be called, if desired, by pressing one of three extra keys on the keyboard, provided jump addresses are provided in certain memory locations.

Not only have Rockwell provided this extensive monitor, but they also make certain software available as extra ROM sets, which plug into the additional sockets on the main computer board. At present two sets are available, one of two ROMs containing an 8K BASIC interpreter, and the other consisting of one ROM contains a two-pass Symbolic Assembler. Either one, or both, of these ROM sets can be installed, since the Assembler and the BASIC interpreter occupy different memory areas.

The version of BASIC supported is that supplied by MICROSOFT, and gives full floating point calculation capability. It is similar in its essentials to the BASICs available on such machines as the APPLE, TANDY TRS 80 and PET. The small on-board random-access capability is a little limiting, but 4K bytes should be sufficient for the types of application that the AIM-65 would usually be used. It certainly cannot be recommended that you try to run BASIC on a machine with only 1K byte of RAM. A useful feature is that when BASIC is entered, the interpreter asks the user what memory area is available for its use for program and data storage.

The rest of memory is then available to the programmer to use for assembler subroutines etc. A comprehensive BASIC manual is provided when the ROMs are purchased, which describes the facilities available.

Although the Symbolic Assembler is provided separately, a fairly extensive Text Editor is included in the Monitor ROM. This is primarily a Line Editor, with commands added to search for the next occurance of a particular string, and replace it, if desired. The Editor forms an integral part of the Assembler package, as it is required to enter and maintain the source program before it is assembled.

On entering the Text Editor for the first time the memory area in which the source text is to be stored must be defined. The program text can then be input from the keyboard, cassette tape, the teletype paper tape reader, or a user defined device. Once entered the text can be manipulated using the Editor Commands. Line numbers need not be stored within the text.

The pointer can be set to the top, or the bottom, line, and it can then be moved up, or down, one line at a time. Lines can be added before the current line, or it can be deleted. Files can be merged, if necessary, by reading the file in with the pointer set to the right point in the text. The currently active line can be displayed, or any desired number of lines can be listed from the current pointer position. Should the Editor be left for any reason it can be reentered without clearing the text buffer by using the warm-entry command. In this way assembler programs can be developed completely within the computer's main memory.

The two string orientated commands allow the user to search for the next occurance of a given string, and either change it, or simply make that line the current line. The string change command has an interesting facility in that when the Editor finds the next line containing the search string it displays it and waits for confirmation. If the line is the one which requires changing the programmer replies by pressing the 'return' key and the line is changed. If it is not the line which requires changing any other key is pressed, and the Editor continues the search, stopping at the next line in which the string occurs.

This means that the whole text can be scanned for a particular string with no fear of changing the wrong line.

The Symbolic Assembler is of conventional two-pass operation. During the first pass a Symbol Table is built up in which all the symbols used are stored, together with their values. The Object Code and program listing are produced during the second pass. The Assembler uses the usual 6502 mnemonics and formatting conventions and is initiated by using a special Monitor command. The Source Code can be taken directly from the Text Editor buffer in memory, or from cassette tape, having been saved there by the Editor. The Object Code can be stored in memory, at the desired address, or may be stored on cassette tape for later use.

If both the source of the program and the Object Code are to be stored on cassette tape two recorders are required. This is necessary for programs of any considerable length, since there is not enough room to store both the source and object codes in the 4K bytes of memory available. The Assembler uses an area of Random Access Memory as workspace, to hold the Symbol Table, and again, this must be allocated by the programmer before assembly begins. It must not conflict with the source and object code buffers, if these are held in memory. It is advisable, therefore, to organise the memory allocation of all the workspaces required before coding commences so that the most efficient use of memory is achieved.

No Macro or Conditional Assembly facilities are provided, but these can hardly be expected on a machine of this size. The arithmetic operations allowed on data items in the Address Field of the instruction codes are also very limited. Of the normal arithmetic and logical operations only addition and subtraction are supported, together with special operators to select either the high, or low byte of a sixteen-bit address.

I have not, in fact, found this a real limitation since, in practice, the usual reason for using the other arithmetic operators is to split addresses up into individual bytes for storage so that they can be used later for indirect jumps etc. It would have been useful to have the logical operators AND, OR and NOT, but these can be simulated with those provided. After all the Assembler Source is not the place to engage in complicated arithmetic computation. Numerical data can be entered in Binary, Octal, Decimal and Hexadecimal formats.

Rockwell have, very sensibly, included the Text Editor and Assembler documentation in the AIM-65 User Manual. Not only is reference material included in this comprehensive manual, but there is also considerable tutorial material both on the use of the facilities provided on the AIM-65, and on 6502 programming in general. There is also an advanced tutorial on the use of the 6522 Versatile Interface Adaptor, a very useful peripheral controller, which also includes two timer circuits. Both the hardware and software are described in detail in the reference sections.

I have used the AIM-65 both for the development of 6502 programs, and for the teaching of Assembler Programming for some time now. Students appreciate the two-pass Assembler in ROM, where it is always available without the time-consuming necessity of loading it in from cassette tape. They are introduced to the Assembler in the first practical, and use it for all their programs. There is little temptation to use either the Mnemonic Instruction Entry facility, or to enter to program as hexadecimal Object Code.

The Disassembler can be used to check that programs have not been overwritten, without resorting to the interpretation of hexadecimal dumps. The printer is also much appreciated since it allows the student to take a listing of his program away with him to study before the next practical session. For most student practicals tape recorders are unnecessary, since the programs written are in general, very short.

It is better to avoid the complexity of extra equipment at what is usually an early stage in their microcomputer programming career. They are, of course, essential for advanced practicals and programming projects when more complex programs are developed over an extended period. The AIM-65 can be used successfully to develop programs not only for itself, but also for other 6502 based microcomputers, such as the PET, APPLE or ACORN. The resulting Object Code must usually be retyped into the target microcomputer, or PROM programmer, unless a suitable teletype interface is available.

To be used as a personal or home computer the AIM-65 really needs to be fitted into a box, with a suitable power supply. Apart from protecting the mechanical and electronic components from dust, dirt, probing fingers etc. it makes the computer much more manageable.

Although the cost of the complete package is comparable with that of a minimum configuration PET or TRS-80 different facilities are offered. It would cost several hundred pounds, for instance, to connect even the cheapest printer to on of the other machines.

Also it is possible to connect a wide range of peripheral devices directly to the AIM-65 through either the peripheral, or the expansion conmnnectors. The AIM-65 can also be used in conjunction with a cheap controller-type single-board computer, such as the ACORN to develop a number of useful projects around the home.

LISTING–1

An Example
of an AIM–65
Assembler Listing

```
<N>
ASSEMBLER
FROM=800   TO=8FF
IN=
LIST?M
LIST-OUT=N

OBJ?
<N>
ASSEMBLER
FROM=800   TO=8FF
IN=M
LIST?Y
LIST-OUT=2

OBJ?N
PASS 1

PASS 2


==0000
       *=$10C
==010C
4C0002 JMP F1
==010F OUTPUT=$E97A

==010F CLR=$EB44

==010F
       *=$200
==0200 F1
2044EB JSR CLR
A2FF   LDX #$FF
==0205 LOOP
E8     INX
BD1402 LDA MSG1,X
C93B   CMP #';'
F006   BEQ RET
207AE9 JSR OUTPUT
4C0502 JMP LOOP
==0213 RET
60     RTS
==0214 MSG1
554C    BYTE 'ULCL M
ICRO'
204C    BYTE ' LAB;'
        END
 ERRORS= 0000
```

TABLE 1

AIM 65 MONITOR COMMANDS

MAJOR FUNCTION ENTRY COMMANDS

(RESET)	— Enter and Initialize Monitor
E	— Enter and Initialize Editor
T	— Re-enter Text Editor at Top of Text
N	— Enter Assembler
5	— Enter and Initialize BASIC Interpreter
6	— Re-enter BASIC Interpreter

INSTRUCTION ENTRY AND DISASSEMBLY COMMANDS

I	— Enter Mnemonic Instruction Entry Mode
K	— Disassemble Memory

DISPLAY/ALTER REGISTER COMMANDS

*	— Alter Program Counter
A	— Alter Accumulator
X	— Alter X Register
Y	— Alter Y Register
P	— Alter Processor Status
S	— Alter Stack Pointer
R	— Display Register Values

DISPLAY/ALTER MEMORY CONTENTS

M	— Display Specified Memory Locations
SPACE	— Display Next 4 Memory Locations
/	— Alter Current Memory Locations

LOAD/DUMP MEMORY COMMANDS

L	— Load Object Code into Memory
D	— Dump Memory

BREAKPOINT MANIPULATION COMMANDS

#	— Clear All Breakpoints
4	— Toggle Breakpoint Enable
B	— Set/Clear Breakpoint Address
?	— Display Breakpoint Addresses

EXECUTION/TRACE CONTROL COMMANDS

G — Start Execution of User's Program
Z — Toggle Instruction Trace Mode
V — Toggle Register Trace Mode
H — Trace Program Counter History

CONTROL PERIPHERAL DEVICES

CTRL PRINT — Toggle Printer On/Off
PRINT — Print Display Contents
LF — Advance Printer Paper
1 — Toggle Tape 1 Control On/Off
2 — Toggle Tape 2 Control On/Off
3 — Tape Verify Block Checksum

TABLE 2

AIM 65 TEXT EDITOR COMMAND SUMMARY

ENTER AND EXIT EDITOR COMMANDS

E — Enter and Initialize Editor
Q — Exit the Text Editor and Return to Monitor

LINE ORIENTED COMMANDS

R — Read Lines into Text Buffer from Input Device
I — Insert One Line of Text Ahead of Active Line
K — Delete Current Line of Text
U — Move the Text Pointer Up One Line
T — Move the Text Pointer to the Top of the Text
B — Move the Text Pointer to the Bottom of the Text
L — List Lines of Text to Output Device
SPACE — Display the Active Line

STRING ORIENTED COMMANDS

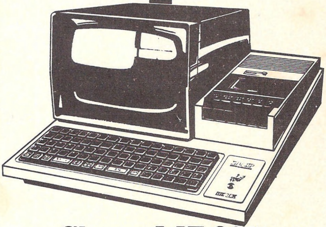
F — Find a Character String
C — Change a Character String

Table 3

Assembler Pseudo Operations

= — Assigns the value of an cper and containing no forward references to either a symbol or the location counter.
.BYTE — Assigns multiple ASCII strings or expressions to consecutive single byte memory locations in high-byte, low-byte order.
.WORD — Assigns multiple expression operands to consecutive memory locations in low-byte, high-byte order.
.DBYTE — Assigns multiple expression operands to consecutive double byte (16 bits) memory locations.
.PAGE — Generates a title under a dashed line.
.SKIP — Generates one blank line.
.OPT — Controls assembly listings. All are optional and can be specified in any order or in separate statements.
.FILE — Last record in a multiple file source program (except the last file) which points to the continuation file.
.END — Last record in a single or multiple source file.

Dare you compare!



Sharp MZ 80K

NOT A KIT
Works the day you buy it
JAPANESE
The same quality they have put into cars and Hi-Fi
SINGLE UNIT
No trailing leads and wires
Z80
More registers and instructions than other processors.
TAPE BASIC
You don't get left with obsolete ROMs.
TAPE COUNTER
Know where you are on the tape.
SOUND
Built-in music synthesiser with 3 octaves.
FAST LOADING
Cassette interface runs at 1200 bps.

Other features — 79 keyboard, up to 48K RAM, on screen editing, real time clock, 256 different characters, 10 inch video display 80 x 50 bit mapped graphics.

DISK DRIVE AND PRINTER COMING SOON

Memory after loading 14K Basic	Nett	VAT	Total
6K	520.00	78.00	598.00
10K	540.00	81.00	621.00
18K	620.00	93.00	713.00
22K	640.00	96.00	736.00
34K	740.00	111.00	851.00

All prices include courier delivery within UK.
Access and Barclaycard Welcome.

MICRODIGITAL

25 Brunswick Street, Liverpool L2 0BJ
Tel: 051-236 0707 (24 hour Mail Order)
051-227 2535 (All other Depts.)
Mail orders to: MICRODIGITAL LIMITED,
FREEPOST (No Stamp Required) Liverpool L2 2AB.

Figure 1
Aim – 65 MEMORY MAP

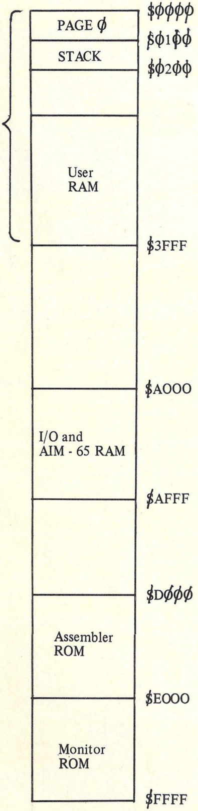
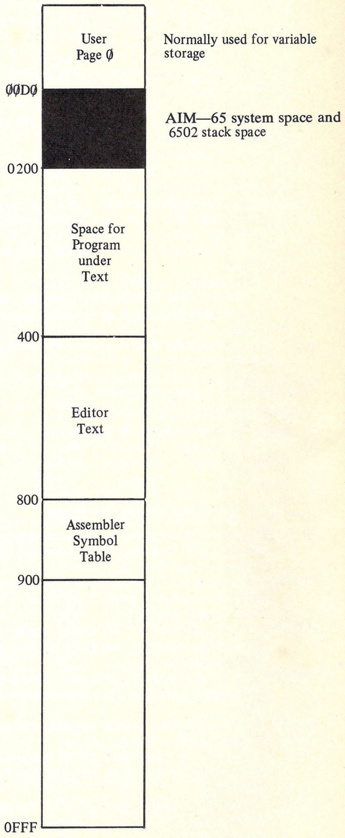

PAGE 0
STACK
User RAM
I/O and AIM - 65 RAM
Assembler ROM
Monitor ROM
RAM
$0000
$0100
$0200
$3FFF
$A000
$AFFF
$D000
$E000
$FFFF

FIGURE 2
A suggested Use of RAM on the AIM–65 when string the 2–Pass Assemblers


User Page 0
Normally used for variable storage
00D0
AIM—65 system space and 6502 stack space
0200
Space for Program under Text
400
Editor Text
800
Assembler Symbol Table
900
0FFF

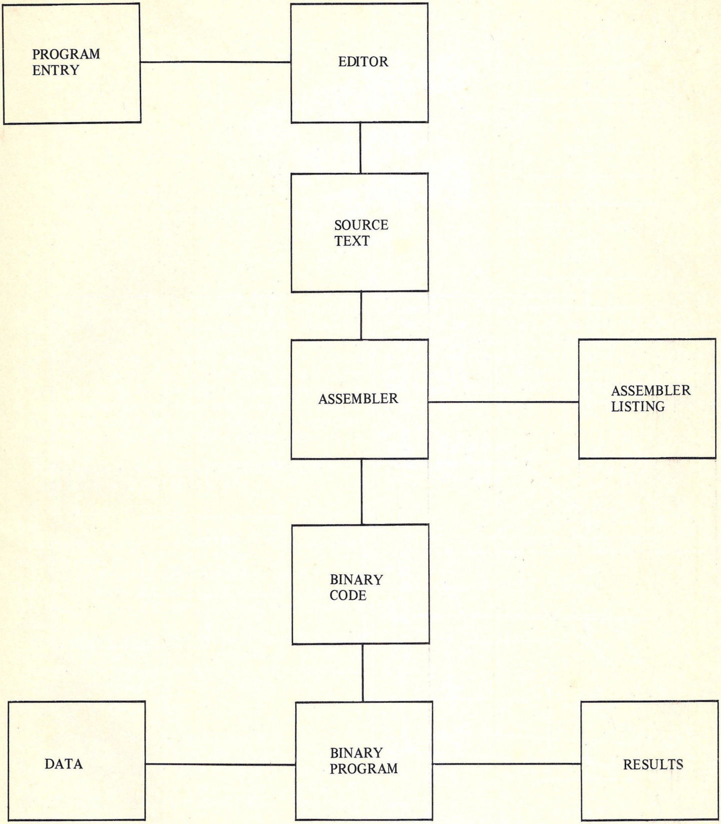

PROGRAM ENTRY
EDITOR
SOURCE TEXT
ASSEMBLER
ASSEMBLER LISTING
BINARY CODE
DATA
BINARY PROGRAM
RESULTS

FIGURE 3

ENTERING AND ASSEMBLING A PROGRAM

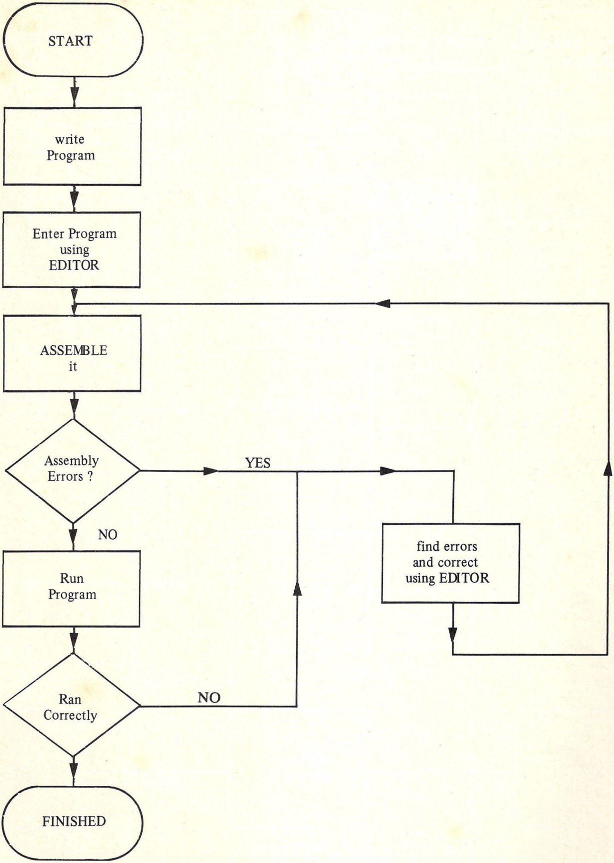
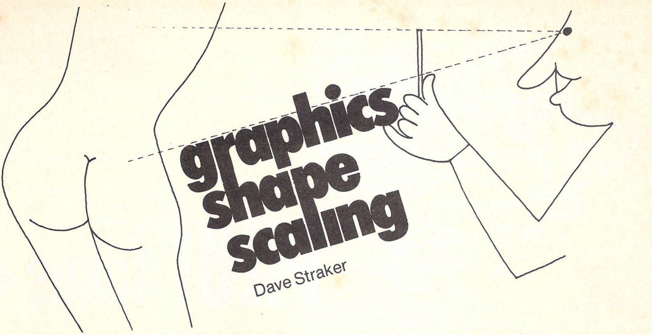

START
write
Program
Enter Program
using
EDITOR
ASSEMBLE
it
Assembly
Errors ?
YES
NO
find errors
and correct
using EDITOR
Run
Program
Ran
Correctly
NO
FINISHED

FIGURE 4

DEVELOPING AN ASSEMBLER PROGRAM



graphics shape scaling

Dave Straker

IN computer graphics separate shapes are often used, such as for drawing spaceships in star-games. It is desireable to be able to make the shape reduce or expand to give the impression of travelling away from or towards the V.D.U. Screen, (Fig. 1). This article describes the theory of how to do just this. It assumes shapes are drawn as successions of points upon the display device.

In order to scale a shape, all points on that shape must be moved towards or away from a given origin in an equal ratio.

Hence shape ABCD in Fig. 2 reduces towards origin O, and becomes A'B'C'D' such that

$$\frac{OX'}{OX} = K$$

where X is any point on ABCD and X' is the corresponding point on A'B'C'D'.

K is the constant scaling factor such that: if K is less than 1, then reduction occurs; if K is equal to 1 then no size change occurs; if K is greater than 1 then expansion occurs.

Each point may be described as a cartesian (x,y) value from the origin, and the co-ordinates (x', y') of the corresponding position are divided in the same ratio.

Thus, considering one point, P, in Fig. 3:

$$K = \frac{OP'}{OP} = \frac{X'}{X} = \frac{Y'}{y}$$

gives the new co-ordinates as (Kx, Ky).

To avoid propagated errors, all scaling should be done from the original shape. Thus the Nth position of P is given by (K1 x K2 x K3 x . . . xKn) x, (K1xK2xK3x . . . xKn)y), where K1 ,K2 . . . Kn are successive scaling factors . . .

With constant scaling factor, K, this position will be $(K^n x, K^n y)$

End stops must be catered for i.e. when the shape expands off the screen, or reduces to a point.

With expansion, the points will separate, and a line will become a series of dots. This may be countered either by drawing lines between adjacent points, or by ensuring that the scaling factor does not exceed 1.

Motion on the screen of the shape is very simple to implement, as the shape is being redefined, it can easily be redefined in any position by moving the origin to that position—i.e. move the origin and the rest of the shape will follow! Various methods of moving shapes will be described in a following article (publishers willing!)

Fig. 1
scaling of a triangle
to give apparent motion
in the z-direction

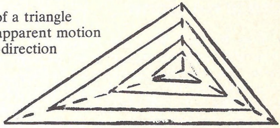


Fig. 2
scaling of shape ABCD

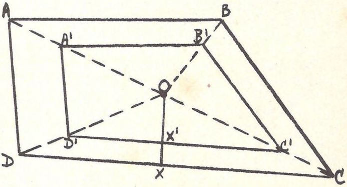
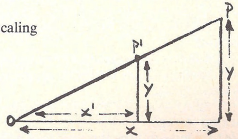
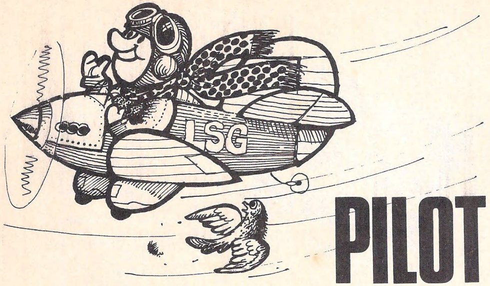


Fig. 3
co-ordinate scaling





Dave Straker

PILOT takes-off

INTRODUCTION

IN my last article on PILOT, I outlined the language as a simple-to-use method of writing text—orientated educational programs. Now teachers (and would-be teachers) can write programs in PILOT using the BASIC program that I have written below.

DESIGN PHILOSOPHY

Why BASIC? Why not Assembler? After all, an Assembler implementation would be both smaller and faster! The obvious question must be answered. BASIC is a language that is commonly available, both on micros and in schools (often on time-sharing terminals linked to a county mainframe). Hence, in order to propagate the language further, it must be in the most useable form. The original was written in Applesoft, and thus has BASIC instructions unique to that particular dialect. In order to facilitate translation to your particular version Table 1 give equivalents to several other BASICS, also remarks in the program Apple-specific sections.

The program is deliberately written as an understandable, modifiable, extendable unit. Commenting is fairly thorough within the program. In an implementation, this may be extracted to shorten and speed the system. Structuring has been used, with major sections starting at a new number group, thus new routines may be added and existing ones changed or removed.

In operation the base level is menu-driven. i.e. a selection is made from a displayed 'menu' and a branch made to the relevant subroutine. These are:

(a) Editor—allows PILOT programs to be input and modified.

(b) Runner—runs the PILOT program interpretively.

(c) Loader—loads a PILOT program from magnetic backing storage.

(d) Saver—saves a PILOT program onto magnetic backing storage.

(e) Ender—returns the user to BASIC.

THE EDITOR

numbering: PILOT lines are not numbered, as in BASIC, but the editor still needs to be able to refer to individual lines, so it numbers the lines by itself—the first being 1), the second 2) etc. Note the ')' used as a separator, hence

12) T:HELLO

is the 12th line, which is T:HELLO. The line number is also used as a prompt, so the request for line 13 is

13)

NOTE that if a line is inserted between lines 12 and 13, then this will now be line 13, the old line 13 will be 14, the old line 14 will be 15 etc. To overcome the loss of lines due to confusion, editing from the bottom up is recommended.

Editor commands are prompted with 'E?', and are:

INPUT NEW PROGRAME, I

typing I will elicit the response:

NEW PROGRAM, OK?

this allows for accidentally hitting the wrong key, as this command erases any existing programs.

NO, N etc. will return control to Editor command level

YES, Y or simply < return > will start input with the first line prompt:

1)

Lines are now input as prompted. A line may be 80 characters long, lines with detected errors are reprompted.

To indicate the end of the program and return to Editor command level, type '#'.

LIST LINES: L m-n

This is similar to the BASIC list instruction, e.g.

L: will list all lines
L-12 will list lines 1 to 12
L5-: will list from line 5 to the end

If a line beyond the given line is used, then the actual line limit of the PILOT program is used.

There is an extra bonus for APPLE users—paddle 0 is used as a list speed controller. At zero setting, the listing will halt, and rotating it clockwise will increase the speed at which lines are displayed.

DELETE LINES: Dm-n

This deletes lines (specified in the same manner as 'list' above) and renumbers lines to maintain the incremental order within the editor, hence

```
1) T: HOW ARE YOU?
2) T:OK?
3) A:
```

—with the command D2 become

```
1) T: HOW ARE YOU?
2) A:
```

APPEND FROM LINE Am

This allows insertion of any number of new lines after line m. Each new line will be asked for with a prompt. Existing lines will be renumbered to accommodate the new lines. To return to Editor command level, just type (return). Note: if # is typed, this makes this the final line in the program, deletes all lines after it and returns to Editor command level. To insert before line 1, append from 0, A0.

CHANGE LINES: Cm-n

This performs a delete plus append. Specified lines (as in 'list' above) are replaced with new lines. See rules for append for inputting new lines.

SEARCH FOR STRING: S string

To facilitate locating lines you may search using any given sequence of characters within that line, so

```
SHELL
```

will give with the first line with HELL somewhere in it, e.g.

```
35) M:HELL! PURGATORY
```

if the end of the program is reached without finding a match,

```
*END OF PROGRAM*
```

is displayed.

The search is cyclic, so if S is again typed, the search will start again from line 1.

QUIT: Q

This returns control to the menu mode.

THE RUNNER

To run a current program type R whilst in menu mode.

When the program finishes, the line, n, it stopped at is displayed in the message.

```
***** PILOT LANDED AT LINE n *****
```

To return to menu mode afterwards, simply hit <return>.To stop at program whilst it is running, type <CTRL-L>.

THE LANGUAGE

STATEMENT LAYOUT

This is of the form:

<instruction> <modifier> : <operand>

<instruction> a single letter indicating instruction type, e.g. Type is T.

<modifier> an optional (signified by the square brackets) boolean condition, which if false, causes this instruction to be ignored. There are two types of modifiers.

(a) Once a numeric variable has been defined in a compute statement, it may be tested against any number in round brackets in the modifier, i.e.

```
T(X < 6):WELL DONE!
```

Types WELL DONE! only if X is less than 6. Other conditions that may be tested are equal (=), less than (<), or greater than (>).

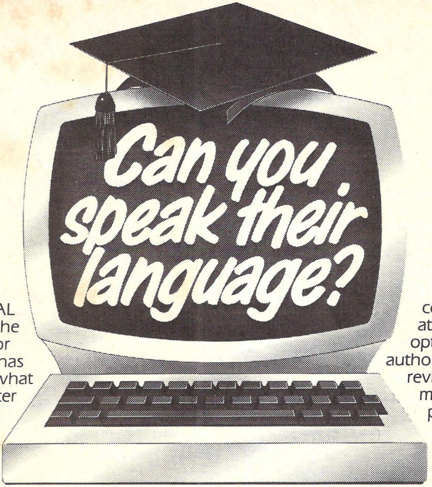
(b) When a match statement, M, is performed a flag is set if a match is found, and reset if no match is found. This flag may be tested with Y (true if flag is set) or N (true if flag is reset), i.e.

TN:WRONG AGAIN... types WRONG AGAIN... only if the last match was unsuccessful.

<operand>= described under each instruction heading.

LABELS

These have '*' as the first character. They are referenced without the '*' in Jump and routine (U) instructions.


Can you speak their language?

EDUCATIONAL COMPUTING is the new magazine for everybody who has ever wondered what the microcomputer revolution has meant to education. Here's your chance to find out everything about the use – and study – of computers and their peripherals in schools, colleges and universities.

Computer services have an obvious part to play in the educational process itself. As well as providing important opportunities for your students to learn many skills which could be vital to their futures, you will benefit personally by greater familiarity with the one subject that's undeniably changing everyone's life.

This unique magazine will offer penetrating investigations into the educational applications and implications of data processing systems; full details of available courses, examinations, and career options; and an authoritative range of reviews – of equipment, software, programs, and books currently on the market.

And our "Beginners' Guide" really does start right at the beginning, with advice on raising the finance for your hardware, and how to be sure you're choosing the right system for your needs.

Teachers and students alike will want to use EDUCATIONAL COMPUTING to keep them abreast of the latest technological developments. As one of the ECC family of top computer publications, you'd be right to expect a knowledgeable – and entirely unpartisan – editorial approach. This is a complex and challenging new field, where there is always something new to learn. So don't risk being left behind. Send the coupon today.

Please enter my subscription for ten issues at the annual rate of £5. I enclose a cheque/PO.
Please charge my credit card (delete as applicable).
Name .... Title ....
School/College/University ....
Address ....
....
.... Signature ....
Type of card .... No: ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐
Send to: EDUCATIONAL COMPUTING, ECC Publications Ltd., 30/31 Islington Green, London N1 8BJ.

e.g.
```
J:FRED
  :
  :
*FRED
```

INSTRUCTIONS

TYPE T:

Prints the operand text. Extra formatting may include:

(a) accepted string variables of the form:

<single character>$

This allows such as Christian names to be used to deformalise programs e.g

```
T:HELLO, WHAT'S YOUR NAME?
A:X$
T:HELLO X$
```

(b) Screen clear. If the first two characters are

!H

e.g.

```
T:!H   ** CHAPTER 2 **
```

MATCH M:

Searches the last Accepted string (see below) for a match with the operand, and sets the match flag, if successful. Special characters are '&' and '!', logical operators 'AND' and 'OR' respectively. e.g.

```
M:NIT&IDE!NITRO
```

This will successfully match with NITRIDE, NITRIC OXIDE, NITRO GEN, etc.

It will fail to match with NITRATE, NITRE, NITRITE, etc.

ACCEPT A:

Halts the program and prompts the user for an input with '?', i.e.

```
A:
```

will give '?' and wait for a string input, which may then be checked with a match statement.

A string variable may be used (see Type, above) and the reply is assigned to that variable, e.g.

```
A:P$
```

COMPUTE C:

Note that this is not a full compute statement—it is only for loop and modifier control. It assigns a number to a numeric variable or increments or decrements a variable by a given amount—thus the formats allowed are:

(a) C:<numeric variable>=<number>

e.g. C: X = 7

(b) C: <numeric variable> = <same variable> + <number>

(c) C: <numeric variable> = <same variable> - <number>

e.g. C: X = X-5

(note that C:P = R+1 will increment P, not R)

JUMP J:

This causes a jump, conditional if modifiers are used, to the named label line e.g.

```
J(P =3): NEXT
        :
        :
        :
*NEXT
```

causes a jump to *NEXT is the numeric variable P is equal to 3.

SUBROUTINE JUMP U:

This is used like jump J:, only a return to the next instruction line may be made with an E: instruction.

SUBROUTINE RETURN E:

There is no operand for this instruction. Execution continues on the line following the last U: instruction encountered.

e.g.

```
  T:THIS
  U:NEXTONE
  T:FUN
  :
  :
*NEXTONE
  T:IS
  E:
```

this prints

```
THIS
IS
FUN
```

REMARK R:

This is ignored by the interpreter, and is for the programmer to comment his listing e.g.

```
R:THIS SECTION DEALS WITH HORSE-PLAY
```

QUIT Q:

The pilot interpreter will halt at the last line of the program. This instruction allows termination at any point within the program e.g.

```
Q(C < 0):
```

a. PROGRAM OPERATION

The program is divided thus

0 - :dimensioning and menu selector
10000 - :Editor
20000 - :Interpreter
30000 - Saves Routines
40000 - Load Routines
50000 - End

The Variable Table gives the variables used. The dimensioning on lines 110 to 140 arbitrarily set a maximum of 100 PILOT lines, with 20 labels, 20 numeric and 20 string variables, and a stack size of 10. Lines 210—320 format the menu. The reply is returned by a get causing an immediate jum via lines 410—450. An invalid input will cause the menu to blink and reprompt.

The Editor gives all commands available, in line 10110, which is displayed whenever an invalid input calls through to line 10350. As List, Change and Delete directions have the same operand, they are processed together. Lines 12102—12330 sort out the lines given, M to N, and validate them. 12510—12590 list the given lines, with 12540—12560 providing Apple:specific control. 12710—12770 delete the lines by shifting up the lines below, and reassigning P1. 12810—12830 changes line by deleting then inputting. 13030—13550 appends from the given line by inputting a line using the input routines of 14300, then shifting lower lines down one to make space for the new line. 14020—14940 inputs lines using 3 routines. 14110—14260 is the control loop, 14330—14560 inputs a line into E$ and does simple syntax checks. 14810—14940 replaces 'INPUT E$' to allow colons and commas, which otherwise are taken as data separators. Control characters are checked far in 14830 (return) 14850 (ct1-X delete line) and 14880 (lift arrow delete last character) no others are supported, but extra routines may be added. 15010—15150 are all the error messages used by the editor and the interpreter. 17100—17200 is the string search routine.

The actual Interpreter starts by scanning the program for labels, and recording their name and line position, lines 200300—20090. 20100—20130 initialises variables. Each interpreted line starts at 20230. 20230—20270 check for non-interpreted lines (remarks, label and end) and break. 20300. 20620 checks the modifier, and if false jumps to the next line. 20700—20820 routes to the appropriate instruction routien. 21000—21130 types are checking for screen clear. 21012—21018 and replacing string variables with the actual text, 21040—21110. 22000—22690 matched. This divides into sections, first by '!', 22120, then by '&', 22400—22490 matches. When a minimum section is reached, A$ is searched for a match, 22600—22690 23000—23320 accepts, and assigns to a string variable if required, 23150—23320. 24000—24210 jumps to line. 25000—25130 jumps to subroutine after saving the current line on the stack, S. 26000—27420 computes by finding the variable, 27110—27260, then setting or incrementing the variable 27300, 27420. 29000—29230 is a routine that removes spaces from the end sof a string, and reduces interior spaces to one space. 30000—save P1 and P$(1) to PS(P1) onto magnetic media. 40000—reverses this. 50000—50040 clears and end the BASIC program.

VARIABLE TABLE

A$	accepted string
C$	general purpose left half of string
D$	general purpose right half of string
D1	'crunch' routine flag
E$	inputted string
E1$	'got' string
H$	string of allowed instructions
I, J, K	general purpose string variable
L(n)	label table line number
L$(n)	label table string number
L1	label table end pointer
N	Editor line number and general purpose
P	current program line
P$(n)	PILOT line table
P1	end of PILOT program pointer
Q(n)	numeric variable value table
Q$(n)	numeric variable name table
Q1	end of numeric variable table pointer
S(n)	stack
S1	stack pointer
V$(n)	string variable name and value table
V1	end of string variable table pointer
X, X$, Z$	general purpose variables

CONVERSION TABLE

APPLESOFT	DESCRIPTION	ALTERNATIVE
LEFT $(X$,Y)	The Y leftmost characters of X$	X$ [1, Y]
RIGHT$ (X$, Y)	The Y rightmost characters of X$	X$ [LEN(X$)-Y, LEN (X$)]
MID$ (X$, Y)	The rightmost characters of X$, from Y	X$ [Y, LEN (X$)]
MIDS (X$, Y, Z)	Z characters of X$, from Y	X$ [Y, Y + Z]
PEEK (-16384)	strobe keyboard, return value of 127 + key value if key is pressed	
POKE—16386,0	resets keyboard strobe	
PDL(X)	returns value 0-255 depending upon analogue paddle X.	
HOME	clears screen	FOR I = 1 TO24:PRINT: NEXT I

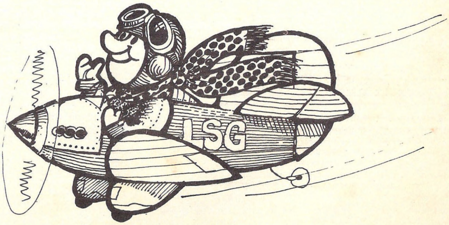
```
]LIST
10  REM **********************************
20  REM ***  PILOT  INTERPRETER        ***
25  REM *** VERSION 1 BY DAVE STRAKER  ***
30  REM **********************************

100  REM *** INITIALISATION ***
110  DIM P$(100),L$(20),L(20)
120 H$ = "TMARJEUCQ"
130  DIM Q(20),Q$(20),S(10)
140  DIM V$(20)

200  REM **********************************
202  REM ***  MENU SELECTOR...          ***
204  REM **********************************

210  HOME : PRINT : PRINT
220  PRINT  TAB( 12);"PILOT INTERPRETER"
230  PRINT  TAB( 12);"================="
235  PRINT : PRINT  TAB( 10);"V1.0 BY DAVE STRAKER "
240  PRINT : PRINT  TAB( 5);"TAKE YOUR PICK..."
250  PRINT
260  PRINT  TAB( 10);"E:EDITOR"
265  PRINT  TAB( 10);"R:RUN PROGRAM"
270  PRINT  TAB( 10);"S:SAVE PROGRAM ON TAPE"
280  PRINT  TAB( 10);"L:LOAD PROGRAM FROM TAPE"
290  PRINT  TAB( 10);"B:RETURN TO BASIC"
310  PRINT : PRINT  TAB( 9);"P?";
320  GET X$
400  REM ** ROUTING TO ROUTINES **
410  IF X$ = "E" THEN  GOSUB 10000
420  IF X$ = "R" THEN  GOSUB 20000
430  IF X$ = "S" THEN  GOSUB 30000
440  IF X$ = "L" THEN  GOSUB 40000
450  IF X$ = "B" THEN  GOSUB 50000
460  GOTO 200

10000  REM **********************************
10010  REM ***  EDITOR'S DESK             ***
10020  REM **********************************

10030  HOME
10100  REM ** INPUT INSTRUCTION **
10102  REM ** I(NPUT NEW PROGRAM)
10103  REM ** L(IST) M,N
10104  REM ** D(ELETE) M,N
10105  REM ** S(EARCH) FOR STRING
10106  REM ** A(MEND FROM) P
10107  REM ** Q(UIT)
10108  REM ** C(HANGE) M,N
10110  PRINT "*****    EDITOR : I,L,D,C,A,S,Q     *****"
10200  PRINT : INPUT "E?";E$: PRINT
10210 C$ =  LEFT$ (E$,1)
10212  IF  LEN (E$) > 1 THEN 10220
10214 D$ = "": GOTO 10300
10220 D$ =  RIGHT$ (E$, LEN (E$) - 1)
10300  REM ** ROUTE COMMAND **
10310  IF C$ <  > "L" AND C$ <  > "D" AND C$ <  > "C" THEN 10320
10312  GOSUB 12000
10314  GOTO 10200
10320  IF C$ <  > "A" THEN 10330
10322  GOSUB 13000
10324  GOTO 10200
10330  IF C$ <  > "I" THEN 10340
10332  GOSUB 14000
10334  GOTO 10200
10340  IF C$ <  > "S" THEN 10350
10342  GOSUB 17000
10344  GOTO 10200
10345  RETURN
10350  IF C$ <  > "Q" THEN 10110
10355  RETURN
12000  REM **LIST,DELETE OR CHANGE LINES M TO N **
12010  REM ** M OR N NULL = START OR END **
12100  REM *SEPARATE M & M *
12102  IF D$ <  > "" THEN 12106
12104 M = 1:N = P1: GOTO 12400
12106  IF  LEFT$ (D$,1) <  > "-" THEN 12110
12108 M = 1:N =  VAL ( MID$ (D$,2)): GOTO 12320
12110  IF  RIGHT$ (D$,1) <  > "-" THEN 12114
12112 M =  VAL ( LEFT$ ((D$), LEN (D$) - 1)):N = P1: GOTO 12310
12114  FOR I = 1 TO  LEN (D$)
12120  IF  MID$ (D$,I,1) = "-" THEN 12180
12130  NEXT I
12140 M =  VAL (D$):N = M: GOTO 12320
12180 N =  VAL ( RIGHT$ (D$, LEN (D$) - I))
12190 M =  VAL ( LEFT$ (D$,I - 1))
12300  REM * CHECK ERRORS IN M & N *
12310  IF M > N THEN 15010
12320  IF M < 1 OR N < 1 THEN 15020
12330  IF N > P1 THEN N = P1
12400  REM * SEPARATE L,D & C *
12410  IF C$ = "D" THEN 12700
12420  IF C$ = "C" THEN 12800
12500  REM * LIST LINES M TO N *
12510  FOR I = M TO N
12520  PRINT I;") ";P$(I)
12530  REM * THIS IS A LIST SPEED CONTROLLER FOR THE APPLE *
12535  REM * SPEED SET BY PADDLE 0 *
12540  IF  PDL (0) < 2 THEN 12530
12550  FOR J = 1 TO 255 -  PDL (0)
12560  NEXT J
12570  REM * END OF SPEED CONTROLLER *
12580  NEXT I
12590  RETURN
12700  REM ** DELETE LINES M TO N **
12710 J = 0
12720  FOR I = N + 1 TO P1
12730 P$(M + J) = P$(I)
12740 J = J + 1
12750  NEXT I
12760 P1 = P1 + M - N - 1
12770  RETURN
12800  REM * CHANGE LINES M TO N *
12810  GOSUB 12700
12815 P = M - 1
12820  GOSUB 13100
12830  RETURN
13000  REM ** CHANGE OR AMEND FROM LINE P *
13010  REM ** NULL STRING TO STOP **
13020 P =  VAL (D$)
13100  REM  *DECIMALISE & ERROR CHECK *
13120  IF P < 0 OR P > P1 THEN 15020
13125 P = P + 1
13130  REM * INPUT A LINE *
13140  GOSUB 14300
13150  IF E$ = "" THEN  RETURN
```

```
13160  REM  MAKE A SPACE AFTR P *
13170  GOSUB 13500
13180 P$(P) = E$
13190  IF C$ = "£" THEN  RETURN
13195 P1 = P1 + 1
13200  GOTO 13125
13500  REM * MOVE IT DOWN A LINE *
13510  FOR I = P1 + 1 TO P STEP  - 1
13520 P$(I) = P$(I - 1)
13530  NEXT I
13550  RETURN
14000  REM *** INPUT NEW PROGRAM ***
14010  REM * '£' ENDS PROGRAM *
14020  INPUT "NEW PROGRAM,OK?";E$
14030  IF  LEFT$ (E$,1) = "N" THEN  RETURN
14040  PRINT
14100  REM ** INITIALISE **
14110 P = 1:P1 = 0
14200  REM * INPUT AND CHECK,LOOP
14210  GOSUB 14300
14225 P$(P) = C$
14230  IF E$ = "" THEN 14210
14240 P$(P) = E$
14245  IF C$ = "£" THEN  RETURN
14250 P = P + 1
14260  GOTO 14210
14300  REM ** INPUT-A-LINE ROUTINE **
14310  REM  * INPUTS E$ *
14320  REM * AND DOES SIMPLE SYNTAX CHECKS*
14330  PRINT P;")";
14335  GOSUB 14800
14340  IF E$ = "" THEN  RETURN
14350 C$ =  LEFT$ (E$,1)
14360  IF C$ <  > "£" THEN 14380
14370 P1 = P: RETURN
14380  FOR J = 1 TO  LEN (E$)
14390  IF  MID$ (E$,J,1) = ":" THEN 14510
14400  NEXT J
14405  IF  LEN (E$) = 1 THEN 15030
14410  REM * NO COLON,SO LABEL CHECK *
14420  IF C$ = "*" THEN  RETURN
14430  GOSUB 15030
14435  GOTO 14330
14510  REM * INSTRUCTION CHECK *
14520  FOR J = 1 TO  LEN (H$)
14530  IF C$ =  MID$ (H$,J,1) THEN 14560
14540  NEXT J
14550  GOSUB 15030: GOTO 14330
14560  RETURN
14800  REM *** REPLACES 'INPUT E$' ***
14810 E$ = ""
14820  GET E1$
14830  IF  ASC (E1$) <  > 13 THEN 14850
14840  PRINT "": RETURN
14850  IF  ASC (E1$) <  > 24 THEN 14880
14860  PRINT "/"
14870 P = P - 1: RETURN
14880  IF  ASC (E1$) <  > 8 THEN 14904
14882  IF  LEN (E$) > 0 THEN 14886
14884 P = P - 1: RETURN
14886  IF  LEN (E$) > 1 THEN 14890
14888 E$ = "": GOTO 14920
14890 E$ =  LEFT$ (E$, LEN (E$) - 1)
```

```
14900  GOTO 14920
14904  IF  ASC (E1$) > 30 THEN 14910
14906  CALL  - 198: GOTO 14820
14910 E$ = E$ + E1$
14920  PRINT E1$;
14930  IF  LEN (E$) > 254 THEN  RETURN
14940  GOTO 14820
15000  REM **** ERROR REPORTER ****
15010  PRINT "*** FOLDBACK ERROR": RETURN
15020  PRINT "*** OUT OF RANGE ERROR": RETURN
15030  PRINT "*** SYNTAX ERROR": RETURN
15040  PRINT "*** DUPLICATE LABEL ERROR": RETURN
15100  REM ** RUN-TIME ERRORS **
15110  PRINT "*** VARIBLE ERROR,LINE ";P: RETURN
15120  PRINT "*** SYNTAX ERROR,LINE ";P: RETURN
15130  PRINT "*** DUPLICATE VARIABLE ERROR,LINE ";P;" & ";I: RETURN
15140  PRINT "*** SUBROUTINE RETURN ERROR,LINE ";P: RETURN
15150  PRINT "*** NO LABEL ERROR,LINE ";P: RETURN
17000  REM *** STRING SEARCH ***
17100  IF D$ = "" THEN 17130
17120 Z$ = D$
17125 P2 = 0
17130  FOR P = P2 + 1 TO P1
17140  IF  LEN (P$(P)) <  LEN (Z$) THEN 17180
17150  FOR I = 1 TO  LEN (P$(P)) -  LEN (Z$) + 1
17160  IF Z$ <  >  MID$ (P$(P),I, LEN (Z$)) THEN 17170
17162  PRINT P;") ";P$(P)
17163 P2 = P
17164  RETURN
17170  NEXT I
17180  NEXT P
17185 P2 = 0
17190  PRINT "*** END FOUND"
17200  RETURN
20000  REM **********************************
20010  REM ***     RUNNER BEAN            ***
20020  REM **********************************
20030  REM *** SET LABEL TABLE ***
20035 L1 = 0
20040  FOR P = 1 TO P1
20050  IF  LEFT$ (P$(P),1) <  > "*" THEN 20090
20060 L1 = L1 + 1
20070 L$(L1) =  MID$ (P$(P),2)
20080 L(L1) = P
20090  NEXT P
20100  REM ** INITIALISE **
20110 V1 = 0:P = 0:M = 0
20120 S1 = 0:Q1 = 0
20130  HOME
20140  IF P1 = 0 THEN 20800
20200  REM ** NEXT LINE LOOP **
20210 P = P + 1
20220 C$ =  LEFT$ (P$(P),1)
20230  IF C$ = "£" THEN 20800
20240  IF C$ = "*" OR C$ = "R" THEN 20200
20250  REM ** STROBE KEYBOARD TO CHECK FOR BREAK (CTL-L) **
20260 X =  PEEK ( - 16384): POKE  - 16386,0
20270  IF X = 140 THEN 20800
20300  REM ** MODIFIER CHECK **
20320  IF  MID$ (P$(P),2,1) = "Y" AND M = 0 THEN 20200
20330  IF  MID$ (P$(P),2,1) = "N" AND M = 1 THEN 20200
20340  FOR I = 1 TO  LEN (P$(P))
20350  IF  MID$ (P$(P),I,1) = ":" THEN 20410
20360  NEXT I
```

```
20370  GOTO 20200
20400  REM * BRACKETS CHECK *
20410  FOR J = 1 TO I
20420  IF  MID$ (P$(P),J,1) = "<" THEN 20450
20430  NEXT J
20440  GOTO 20700
20450  FOR K = J TO I
20460 D$ =  MID$ (P$(P),K,1)
20470  IF D$ = "=" OR D$ = "<" OR D$ = ">" THEN 20500
20480  NEXT K
20490  GOTO 15120
20500  IF Q1 = 0 THEN 15110
20510  FOR N = 1 TO Q1
20520  IF  MID$ (P$(P),J + 1,K - J - 1) = Q$(N) THEN 20545
20530  NEXT N
20540  GOTO 15110
20545 J =  VAL ( MID$ (P$(P),K + 1,I - K - 1))
20550  IF D$ <  > ">" THEN 20580
20560  IF Q(N) > J THEN 20700
20570  GOTO 20200
20580  IF D$ <  > "<" THEN 20610
20590  IF Q(N) < J THEN 20700
20600  GOTO 20200
20610  IF Q(N) = J THEN 20700
20620  GOTO 20200
20700  REM *** INSTRUCTION ROUTING ***
20710 D$ =  MID$ (P$(P),I + 1)
20720  IF C$ <  > "T" THEN 20730
20725  GOSUB 21000: GOTO 20200
20730  IF C$ <  > "M" THEN 20740
20735  GOSUB 22000: GOTO 20200
20740  IF C$ <  > "A" THEN 20750
20745  GOSUB 23000: GOTO 20200
20750  IF C$ <  > "J" THEN 20760
20755  GOSUB 24000: GOTO 20200
20760  IF C$ <  > "U" THEN 20770
20765  GOSUB 25000: GOTO 20200
20770  IF C$ <  > "E" THEN 20780
20775  GOSUB 26000: GOTO 20200
20780  IF C$ <  > "C" THEN 20790
20785  GOSUB 27000: GOTO 20200
20790  IF C$ = "Q" THEN 20800
20800  PRINT : PRINT "   *** PILOT LANDED AT LINE ";P;" ***"
20810  INPUT " ";E$
20820  RETURN
21000  REM *** T: TYPE ***
21010  IF  LEN (D$) < 2 THEN 21120
21012  IF  LEFT$ (D$,2) <  > "!H" THEN 21020
21014  HOME
21016  IF  LEN (D$) = 2 THEN  RETURN
21018 D$ =  MID$ (D$,3)
21020 I = 1
21030 I = I + 1
21040  REM * REPLACE STRING VARIABLES *
21050  IF  MID$ (D$,I,1) <  > "$" OR V1 = 0 THEN 21110
21060  FOR J = 1 TO V1
21070  IF  LEFT$ (V$(J),2) <  >  MID$ (D$,I - 1,2) THEN 21100
21072 E$ =  MID$ (V$(J),3)
21074  IF  LEN (D$) <  > 2 THEN 21076
21075 D$ = E$: GOTO 21120
21076  IF I <  > 2 THEN 21080
21078 D$ = E$ +  MID$ (D$,I + 1): GOTO 21100
21080  IF  LEN (D$) <  > I THEN 21084
21082 D$ =  LEFT$ (D$,I - 2) + E$: GOTO 21100
21084 E1$ =  LEFT$ (D$,I - 2) + E$
21086 D$ = E1$ +  MID$ (D$,I + 1)
21088  GOTO 21100
21090 I = I +  LEN (V$(J)) - 2
21095  GOTO 21110
21100  NEXT J
21110  IF I <  =  LEN (D$) THEN 21030
21120  PRINT D$
21130  RETURN
22000  REM *** M: MATCH ***
22010  REM * INITIALISE *
22020 M = 0:M1 = 0:M2 = 0
22030  GOSUB 29000
22100  REM * LOOK FOR !'S *
22110  FOR I = 1 TO  LEN (D$)
22120  IF  MID$ (D$,I,1) <  > "!" THEN 22170
22130  REM * GO LOOK FOR &'S *
22140  GOSUB 22400
22150  IF M = 1 THEN  RETURN
22160 M1 = I
22170  NEXT I
22190  GOSUB 22400
22200  RETURN
22400  REM * LOOK FOR &'S IN A$ *
22410 M3 = 0:M2 = M1
22420  FOR J = M1 + 1 TO I - 1
22430  IF  MID$ (D$,J,1) <  > "&" THEN 22470
22440  GOSUB 22600
22445 M2 = J
22450  IF M = 0 THEN  RETURN
22470  NEXT J
22485  GOSUB 22600
22490  RETURN
22600  REM * LOOK FOR MATCH IN A$ *
22610  FOR K = M3 + 1 TO  LEN (A$) - J + M2 + 2
22620  IF  MID$ (D$,M2 + 1,J - M2 - 1) <  >  MID$ (A$,K,J - M2 - 1)
       THEN 2  2670

22640 M = 1
22650 M3 = K + J - M2 - 2
22660  RETURN
22670  NEXT K
22680 M = 0
22690  RETURN
23000  REM *** A: ACCEPT ***
23100  PRINT "?";
23110  GOSUB 14800
23120  IF D$ <  > "" THEN 23150
23130 A$ = E$
23140  RETURN
23150 Z$ = D$:D$ = E$
23200  REM * ASSIGN A$ TO NEXT V$ *
23210  GOSUB 29000
23215 A$ = D$:D$ = Z$
23230  IF V1 = 0 THEN 23300
23240  IF  LEN (D$) <  > 2 OR  RIGHT$ (D$,1) <  > "$" THEN 15110
23250  FOR I = 1 TO V1
23260  IF D$ = V$(I) THEN 15130
23270  NEXT I
23300 V1 = V1 + 1
23310 V$(V1) = D$ + A$
23320  RETURN
24000  REM *** J: JUMP ***
```

```
24010  GOSUB 29000
24100  FOR I = 1 TO L1
24120  IF D$ = L$(I) THEN 24200
24130  NEXT I
24140  GOTO 15150
24200 P = L(I)
24210  RETURN
25000  REM *** U: GO TO SUBROUTINE ***
25100 S1 = S1 + 1
25110 S(S1) = P
25120  GOSUB 24000
25130  RETURN
26000  REM *** E: RETURN FROM SUBROUTINE ***
26100 P = S(S1)
26110 S1 = S1 - 1
26120  IF S1 < 0 THEN 15140
26130  RETURN
27000  REM *** C: COMPUTE ***
27100  IF  LEN (D$) < 3 THEN 15120
27110  REM * FIND "=" *
27120  FOR I = 1 TO  LEN (D$)
27130  IF  MID$ (D$,I,1) = "=" THEN 27160
27140  NEXT I
27150  GOTO 15120
27160  IF Q1 = 0 THEN 27240
27200  REM * CHECK FOR EXISTING N$ *
27210  FOR J = 1 TO Q1
27220  IF  LEFT$ (D$,I - 1) = Q$(J) THEN 27300
27230  NEXT J
27240 Q1 = Q1 + 1
27250 J = Q1
27260 Q$(J) =  LEFT$ (D$,I - 1)
27300  REM * FIND OPERATION *
27310  FOR K = I + 1 TO  LEN (D$)
27320  IF  MID$ (D$,K,1) = "+" OR  MID$ (D$,K,1) = "-" THEN 27400
27330  NEXT K
27335  IF I =  LEN (D$) THEN 15120
27340 Q(J) =  VAL ( MID$ (D$,I + 1))
27350  RETURN
27400  REM * INCREMENT VARIABLE *
27410 Q(J) = Q(J) +  VAL ( MID$ (D$,K))
27420  RETURN
29000  REM *** CRUNCH SQUEEZES SPACES ***
29100  IF  LEN (D$) < 2 THEN  RETURN
29110 D1 = 0:I = 0
29120  REM * LOOP HERE *
29130 I = I + 1
29140  IF  MID$ (D$,I,2) <  > "  " THEN 29170
29150 D1 = 1
29152  IF I > 1 THEN 29160
29154 D$ =  MID$ (D$,2): GOTO 29130
29160 D$ =  LEFT$ (D$,I - 1) +  MID$ (D$,I + 1)
29170  IF I <  >  LEN (D$) THEN 29130
29180  IF D1 = 1 THEN 29110
29190  IF  LEFT$ (D$,1) <  > " " THEN 29210
29200 D$ =  MID$ (D$,2)
29210  IF  RIGHT$ (D$,1) <  > " " THEN  RETURN
29220 D$ =  LEFT$ (D$, LEN (D$) - 1)
29230  RETURN
30000  REM ********************************
30010  REM ***          SAVIOUR          ***
30020  REM ********************************
30030  HOME
30100  PRINT "*****          SAVER          *****"
30110  PRINT : PRINT
30120  REM * PUT YOUR ROUTINE HERE TO SAVEP1,(END OF PROGRAM POINTER),AND
    P$ (PROGRAM ARRAY) *
39900  RETURN
40000  REM ********************************
40010  REM ***        LOADER           ***
40020  REM ********************************
40030  HOME
40100  PRINT "*****          LOADER          *****"
40110  PRINT : PRINT
40120  REM * PUT YOUR ROUTINE HERE TO LOAD P$ (PROGRAM ARRAY) AND P1 (PROG
    RAM END POINTER) *
49900  RETURN
50000  REM ********************************
50010  REM ***  ARMAGEDDON             ***
50020  REM ********************************
50030  HOME
50040  END
```



Isaac Newton's easy solution!

Apple prices are tumbling at Microdigital!

A 48K Disk based computer system for only **£1044** +VAT For March, April, May and June only.

	Nett	VAT	Total
16K byte Apple Computer	£695.00	£104.25	£799.25
Disk System	£349.00	£52.35	£401.35
Total with 32K free extra memory	£1044.00	£156.00	£1200.60

Latest Apple II plus model with floating point basic and auto-start ROM.

Buy a 16K Apple with Disk drive and get 32K of memory Free.

Extras			
Second Disk Drive	£299.00	£44.85	£343.85
Pascal Language System	£299.00	£44.85	£343.85
Graphics Tablet	£462.00	£69.30	£531.30
Appletel System	£595.00	£89.25	£684.25
Black & White Modulator	£14.00	£2.10	£16.10
Eurocolour Card	£79.00	£11.85	£90.85
Joystick	£25.00	£3.75	£28.75

All Prices include courier delivery within U.K.

MICRODIGITAL

25 Brunswick Street, Liverpool L2 0BJ
Tel: 051-236 0707 (24 hour Mail Order)
051-227 2535 (All other Depts.)

Mail orders to: MICRODIGITAL LIMITED,
FREEPOST (No Stamp Required)
Liverpool L2 2AB.

OFFICAL ORDERS

We Welcome official orders from bona-fide commercial and government organisations. We require payment 14 days after invoice date.

Apple Business Systems -ring Graham Jones 051·227 2535

Software Packages	£
Stock Control	£225.00
General Ledger	£295.00
Purchase Ledger	£295.00
Sales Ledger	£295.00

Word Processor/Letter	£
Writer	£130.00
Book keeper Package	£295.00
Credit Control Package	£150.00
Payroll	£385.00

+ VAT at 15%

CRA

apple computer Sales and Service

Please send systems as above at £1200.60 in addition please send:

Total Remittance ________________

Name ________________

Address ________________

____

Access and Barclaycard welcome.

Letter from America

D.Smith

Editor, THE APPLE SHOPPE
THE JOURNAL OF APPLE APPLICATIONS
SUBSCRIPTIONS: $24.00 A YEAR.
THE APPLE SHOPPE
P.O. BOX 701
PLACENTIA, CA 92670 USA

HELLO to all my friends in England. This month we present a new column that will let you know what is happening here in America so you can keep up with the latest US trends in small systems computing. We welcome your response and hope you will drop us a line and let us know how developments here are affecting you in England. Just send your comments to the above address, and we will try to answer as many as we can through this column. Personal replies only with a self-addressed stamped envelope please. Postage rates from the US to England do add up!

NEW COMPUTERS ARRIVE

Several new computers have arrived on the scene here and are being received with mixed reactions. Many computer dealers now have the new Atari 400's and 800's but so far, the Apple look-alikes have not burned up the marketplace Indications are that the Atari 800 will become a moderately good seller, with the 400 much less so. But the Atari's poor showing in the display window reportedly has helped Apple sales where the two units have been displayed side by side. The Atari is well known for having good shielding, having passed the government FCC regulations for radiation interference. The trouble is, while it may not radiate anything, it picks up everything! So when the Atari is placed next to an Apple, the Apple screen is sharp and clear, while the Atari suffers from worms and waves in its display and really shows the unit off in a bad light. The reported cause of this is the poor quality rf modulator that is built-in to the Atari. The Apple, on the other hand, uses an external modulator, so some freedom of choice is available to choose a good modulator. Still, the Atari is bringing in new customers into the marketplace and that is good for everyone.

TI UNIT SALES DISMAL

Another new unit, the T.I. home computer is having a dismal time and dealers are reporting very few takers for the unit. It appears that T.I. has guessed wrong about the strength of the home computer market, and that most people simply are not willing to spend over $1000 for what amounts to a sophisticated video game. Rumor has it that T.I. will phase out the unit in favor for a new lower cost unit, without the color monitor, now that T.I. has received FCC clearance on their units. Still, both the Atari units and the T.I. unit suffer from a bad price/performance ratio when compared with the Apple or radio shack, because of the relatively high cost of the peripherals and memory expansion.

H.P. HITS THE MARK!

One new unit that is receiving enthusiastic response is the new H.P. 85 personal computer. This new unit is everything the T.I. unit is not. It features a built in video screen, H.P. digital tape unit, thermal printer all in one small desk top unit. While the price of $3200 is a bit high by micro-computer standards, this unit carries all the quality and backing of the H.P. line. Included is an impressive graphics capability and scientific oriented BASIC compiler. The built-in printer prints anything on the screen including graphics, and the unit accepts a wide variety of peripherals including I.E.E.E. 488 or

RS-232 peripherals. It no doubt will become the lowest cost bus controller available, and will probably show up in every engineering department in the country very quickly. Perhaps it will become the pawn in the upcoming press for manpower, with companies offering H.P.85's to attract top engineering talent during the projected manpower shortage in the next few years. Dealers report a very health demand for this unit, with H.P. already on allocation until June.

IBM MAKES ITS MOVE

IBM is making its long expected move into the mini and micro field with their new distributed processing system, the 5120. This unit is actually the same as their 5110 unit which has been on the market for some time, but features a lower price, with a large video screen. The cpu is rumored to emulate a 360 which accounts for the slow speed of the 5110. It is not known if this has been changed in the new unit. With the desk top processor-display unit, are built in dual floppies for 1.2 megabytes on line with hard disk capability optional. Best of all, a full range of small business accounting software is available. Now check the price: $13,000 to $23,000 depending on model and options! This compares very favorably with many small business micro's which are now inching up into the 10 to 20 thousand dollar range with hard disk capability. Now I ask you, if it came to a choice between a North Star 8080 and an IBM computer, which would you choose? My guess is that IBM will take a lot of suffering out of the sails of budding micro business systems, which seem to be proliferating at an unbelievable rate. IBM will market the unit at IBM retail outlets, some 200 which are planned for this year. The unit will be 'off the shelf' although its weight of over 100 pounds may make that a bit impractical!

HARD DISKS ARE IN!

This is the year of the hard disk, with many vendors offering an assortment of hard disk capability for their systems. Corvus is marketing a 10 megabyte 8 inch winchester hard disk for the Apple, TRS-80 and S-100 bus computers. The company claims to have solved the back-up problem with a tape drive unit for around $1500 that will back-up the hard disk in just a few minutes. The Apple version is very impressive running Apple Pascal. The Pascal driver is supplied with the drive and allows the entire 10 megabtyes to be addressed as a single block or broken up into smaller blocks. Complete compatibility is maintained with both DOS 3.2 and with the 280 block Apple Pascal disk format. An Apple pascal system with 10 megabytes on line is a very powerful system indeed and would rival many mini's in processing power for a fraction of the cost. The corvus unit is $5350.

Look for a lot of activity in this area, since Shugart has announced a 2 megabyte hard disk that fits into an 8 inch floppies, why not? It won't take long for the Shugart unit and others from Japan to open up the 1 to 2 megabyte market, which if the cost is down around $2500, will be extensive.

NEW APPLE USERS GROUP

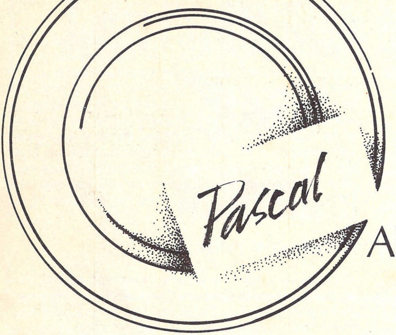
A new Apple users group is being organized as the club-to-end-all clubs! Called the International Apple Core, the club will actually be a user group for Apple user groups! Val Golding of Call—A.P.P.L.E. fame will be editing the club magazine, called the Apple Orchird, which promises to be sent to every Apple owner world wide. Apple has already supplied the new user group with mountains of technical material, which has been sent to member user groups for distribution. Check with your local user group for details on how you can join. Only user groups can actually become members of the International Apple Core.

NETWORKING TAKES OFF

The latest craze here is networking with micros. Two national networks have really caught on with nearly every micro vendor. These are The Source, and Micronet. With these systems, you can call a friend on your computer from L.A. to N.Y. for just $2.90 an hour! This will really open up personal computing. In fact, rumor has it that GTE and the other big telephone companies are aiming to do just that, by getting in on the bandwagon and offering the personal computer telephone that will serve both communication and computing needs in the home. In light of this, some people are projecting a serious manpower shortage in the telecommunications industry for skilled technical people. One of the really neat things about systems like The Source, is that your micro can tie into the huge news gathering services of AP and UPI, not to mention the DOW JONES stock information! Maybe a London—New York link can be established?!

MICRO NEWS

In the personal computing news, look for Programma International to release a new version of Lisa and Apple Pi. The new Apple Pi text editor is supposed to be better than anything on the market including Easywriter. Speaking of Easywriter from Information Unlimited, they have announced a new plug in board for the Apple that gives upper and lower case from the Apple keyboard, and 80 columns display on the screen! It is rumored to be Apple Pascal compatable, but that has not been verified yet. If it is, it means the excellent Pascal editor will be available for text editing without having to buy an external terminal. More on this development next time. Please feel free to write and let me know how things are in England. And I hope you Apple owners will take advantage of my APPLE SHOPPE publication, as I am sure you will find it a great help to you. That's all from America!



AN INTRODUCTION

Dr Andrew Veronis

President ACFA Inc.
P.O Box 1070 Glen Burnie
Maryland 21061
USA

PASCAL is fast becoming one of the most popular programming languages for microcomputers. It is for this reason I developed the urge to write a long series of articles, and to describe this fascinating language.

ELEMENTS OF PASCAL

Pascal, like any other programming language, has its own syntax that determines the way Pascal identifiers and symbols are put together to form statements oriented to Pascal Computational processes.

NOTIONS

The basic Pascal vocabulary consists of keywords plus special symbols used as operators and delimiters. In this series of articles, Pascal keywords are always capitalized. For example, the following are Pascal keywords as they might appear throughout the artciles: WHILE, DO, BEGIN, ARRAY, CASE. An example of syntax is:

```
WHILE expression DO
statement:
```

In syntax, when items are to be replaced with values supplied by the user, these items are lower case, and the name suggests the type of value to be supplied. In the example above, expression and statement are supplied as appropriate by the programmer. An actual program example might look like this:

```
  WHILE N>5 DO
  Begin
  SUM:= SUM + 1/N;

  N: = N-1
END;
```

In syntax, optional items appear enclosed in vertical bars. For example:

```
REPEAT statement /statement/ . . .
UNTIL booleanexpression
```

One or more repetitions of a single syntactical item is indicated by a succession of three periods (. . .) as in the example above.

CONSTANTS

A constant is a literal representation of fixed, unvarying value, associated with some data type. A Pascal constant can be a decimal integer with no decimal point. For example:

```
652
-25
```

A constant can be a character enclosed in apostrophes. For example:

```
'A'
'B'
'$'
```

A constant can be a character string enclosed in apostrophes. For example:

```
'CHARACTER STRING', 'ANOTHER STRING'
```

Note that the range of values for decimal constants is—32768≤decinteger≤32768. Also, character strings must be 255 or fewer in length.

IDENTIFIERS

Identifiers are names chosen by the programmer to denote constants, types, files, variables, procedures and functions. The first character of an identifier must be a letter. Any number of letters and digits can follow. However, identifiers must differ in the first 8 characters to be distinct.

Most of Pascal compilers in use today recognise the set of standard identifiers specified below. These identifiers can be redefined locally or globally:

Standard constants are:

FALSE MAXINT TRUE

Standard types are:

BOOLEAN CHAR INTEGER

REAL TEXT

Standard files are:

INPUT OUTPUT

Standard Functions are:

ABS ARCTAN CHR
COS EOF EOLN
EXP LN ODD
ORD PRED ROUND
SIN SQR SORT
SUCC TRUNC

Standard procedures are:

GET NEW PAGE
PUT READ READLN
RESET REWRITE WRITE
WRITELN

Punctuation symbols

Almost half of the Pascal puncuation symbols (including mathematical operators) are the same as in any order programming language. However, quite a few more have a different meaning, and are described briefly below:

AND Boolean conjunction.
OR Boolean incjusive disjunction.

NOT	Boolean complement
=	becomes (replacement)
,	separates the items in a list
;	separates statements
:	separates variable name and its type
,	delimits character string literals
.	Decimal point, record selector, program terminator
. .	subrange specifier
∧	indicates file or pointer variable
(	starts paramenter list or nested expression
)	end parameter list or nested expression
	starts subscript list or set expression
	ends subscript list or set expression
*)	starts a comment
(*	ends a comment
{	starts a comment
}	ends a comment

RESERVED KEYWORDS

The following are standard Pascal keywords which are reserved and each keyword is considered a distinct special symbol that cannot be used in any context other than in the explicit definition of Pascal:

AND ARRAY BEGIN
CASE CONST DIV
DO DOWNTO END
ELSE FILE FOR
FUNCTION GOTO IF
IN LABEL MOD
NIL NOT OF
OR PACKED PROCEDURE
PROGRAM RECORD REPEAT
SET THEN TO
TYPE UNTIL VAR
WHILE WITH

COMMENTS

Comments can be added to a Pascal program and can be inserted between any too identifiers, numbers, or special symbols. The general format is:

(* *)

Examples:

(* THIS IS A COMMENT*)
{ ANOTHER COMMENT }

When the compiler encounters a left-hand comment symbol, it scans the text for the matching right-hand comment symbol. Do not intermix the two types of comment disignators.

TERMS AND DEFINITIONS

The following terms are often used in place of formal parameters in dummy declaration headers:

ARRAY	A PACKED ARRAY OF CHARACTERS
BLOCK	one disk block (512 bytes)
BLOCKS	an INTEGER number of blocks
BOOLEAN	any BOOLEAN value
CHARACTER	any expression which evaluates to a character
DESTINATION	a PACKED ARRAY OF CHARACTERS to write into or STRING, context dependent.

EXPRESSION	part or all of an expression, to be specified
FILEID	a fil identifier, must be

```
VAR fileid:  FILE OF<type>;
         or:  TEXT;
         or:  INTERACTIVE;
         or:  FILE;
```

INDEX	an index into a STRING or PACKED ARRAY of CHARACTERS context dependent or as specified.
NUMBER	a literal or identifier whose type is either INTEGER or REAL.
RELBLOCK	a relative disk block address, relative to the start of the file in context, the first block being block zero.
SIMPLAVARIABLE	any declared Pascal variable which is BOOLEAN, CHAR, REAL, STRING, or PACKED ARRAY [. .] OF CHAR
SIZE	an INTEGER number of bytes or characters; any integer value
SOURCE	a STRING or PACKED ARRAY OF CHARacters to be used as a read-only array, context dependent or as specified
STRING	any STRING, call-by-value unless otherwise specified, i.e., can be a string variable, or a function which evaluates to a string.
TITLE	a STRING consisting of a filename
UNITNUMBER	physical device number used to determine device handler used by the interpreter

PROGRAM STRUCTURE

Every Pascal program has two required parts, a program heading and a block. The first word of all Pascal programs is 'PROGRAM'. The block contains two main sections:

> Declaration/definition Section specifies all objects local to the program. Consists of five specific parts which are described as the discussion proceeds.
>
> Label declaration part.
> Constant definition part
> Variable declaration part
> Procedure/function declaration part

The State Section spcifies all actions to be performed upon the above declared objects.

PROGRAM HEADING PART

The program heading gives the program a name. For example:

```
PROGRAM ADD;

PROGRAM INVENTORY;
```

LABEL DECLARATION PART

The label declaration part sets up statement labels for reference by GOTO statements. Each label must be an unsigned integer of four digits or less. For example:

```
LABEL 214, 56, 5;
```

CONSTANT DEFINITION

The constant definition part gives program constants a name (identifier) to be used within the program as a synonym. Constants can be numbers, as tring, or another constant. For example:

```
CONST

  ROW LENGTH = 17;
  MAXVAL = 14.5;
  QUESTMARK  '?';
```

If the particular Pascal compiler you are to use employs values that are known before program execution, you can write each value two different ways, as a constant or a literal. Actually a constant is a literal with a name. Constants are given names in the CONST declaration section. The following are program segments using literals:

```
WHILE X<15 DO . . .
FOR INDEX: = 1   TO 25 DO . . .
```

Following are some examples using constants set by CONST:

```
CONST
  MAX = 20;
  HIGH ≈25;
  PI = 3.141593;
WHILE X<MAX DO . . .
FOR INDEX: = 1 TO HIGH DO . . .
```

In the second example, literals are given names in the CONST declaration section which defines the entity to have a name and a value. Well-written programs rarely use literals outside the CONST declaration section. This makes program changes easier.

Each data item in a Pascal program is either a constant or a variable; constants remian the same during program execution, but variables can vary. Constants are declared in the CONST section. Variables are declared in the VAR section. Each data item is of a specified type, and its type determines the kind of operations that can be performed on it. In Pascal, the four standard types of data are INTEGER, REAL, BOOLEAN, and CHAR.

Examples of the four standard data types are:

INTEGER	REAL	BOOLEAN	CHAR
—20	—20.5	TRUE	'A'
15	15.274	FALSE	'2'

When you declare constants in the CONST section, their type is implicit in the declaration and you need not specify the type. When you declare variables in the VAR section, you must specify the type by using INTEGER, REAL, BOOLEAN, or CHAR. For example:

```
VAR

   X: INTEGER;
   MAX, COUNT, AZIMUTH: REAL;
   FLAG, ENDFILE: BOOLEAN;
   SIGNAL: CHAR;
```

Note that numerous variables can be typed at the same time and that the VAR section does not assign actual values to the variables. The actual value of a variable is determined by the execution of an assignment statement of the form:

```
variable: = expression
```

For example:

```
X: = 20;
AZIMUTH: = 75.92 * MAX + COUNT;
FLAG: = TRUE;
SIGNAL: = 'M';
COUNT: = COUNT + 2.4;
```

To review then, data items are either CONSTants or VARiables, and their type can be of one or of the following: INTEGER, REAL, BOOLEAN, CHAR. These are standard Pascal data types; others are possible by explicit definition with the TYPE Statement.

Next month, we will continue with more description of the interesting world of Pascal.

Nascoms Programs & Information

a book by

Merseyside Nascom Users Group

ONLY £2.25

Including P & P

Yours at . . .
Microdigital Limited
25 Brunswick Street
Liverpool
L2 0PJ

SUPPORT MEMBERS OF THE COMPUTER RETAILERS ASSOCIATION . . .

COMPUTER RETAILERS ASSOCIATION
CRA

THEY WILL SUPPORT YOU.

For further details on the associations aims. membership, code of conduct etc.

Please contact: Mrs Gibbons,
Owles Hall, Butingford,
Hertfordshire, SE9 9PL.
Tel. (0763) 71209

Programming Practices and Technics

MICROCOMPUTER PROGRAMMING TECHNIQUES

TEXT EDITORS

Martin D. Beer,
Computer Laboratory
University of Liverpool.
P.O. Box 147
Liverpool,
L69 3BX

Part III

apple
LIVERPOOL SOFTWARE GAZETTE

FROM the earliest days of electronic computers, it was realised that they could be used for the manipulation of textual information. Although the earliest machines were used, primarily for numerical work, they were also required to read, and senerate textual information in the form of program data, and results. The earliest computers were very expensive, and computer time was strictly rationed. No-one would therefore consider the use of precious computer time for the preparation of either programs or data. All this was done offline, and programmers queued up to feed their punched cards, or paper tapes into the computer, and obtain their results. The programs were written directly in machine code, so program changes were difficult, and time consuming to introduce. Only once computers were introduced into the commercial world, was any real consideration given to how the computer could help make the programmer's task easier.

With the introduction of first, assemblers, and later compilers, it became necessary to create, and maintain large amounts of machine-readable textual information, in the form of program sources. At first, these were held either on paper tape, or on punched cards, in the same way as machine-code programs, and data had been before. The program text was ammended by removing the relevent punched cards, and replacing them with new ones holding the required text, or by repunching the whole paper tape, making the necessary corrections, as you went. It was possible, if your computer had a forgiving paper-tape reader, to cut out the sections of tape to be corrected, and splice in new sections, suitably corrected. Although paper tape was more difficult to handle than punched cards, it had the advantages that it was lighter, and more compact, and could not be mixed up, when dropped. Because punched card equipment, such as mechanical sorters, required the data to be held in fixed fields, assemblers, and high-level languages, designed to be used on computers with, primarily, card based input/output required that the programmer used particular columns for different purposes. The computer was very unforgiving, and would reject any card which did not precisely fit the predefined conventions. It was also usual to follow the convention of numbering each card, in a special field, so that if a card deck was dropped, it could be put back together again, in the correct order, with the help of an automatic sorter. The strict field conventions of most assemblers, and some high-level languages, notably FORTRAN, are a throw-back to these days.

As computers became cheaper, and more common, consideration was given to the connection of mass storage devices, such as magnetic tape units, and later, magnetic drums and disks. At first these storage units were used to store the operating system and programs which were run frequently, such as the assemblers and compilers. They were then used to store the programs, and data, temporarily, so that throughput could be improved. It was then realised that the information required could be held permanently on the mass-storage medium, and manipulated using special utility programs. These were the first true text editors. Early editors were able to insert, replace and delete complete lines, which the programmer identified by their line numbers, stored within the text. As computers became yet more powerful, and users amassed considerable quantities of data, the primative facilities then available became woefully inadequate. Editors were developed, which not only allowed the programmer to manipulate individual lines of text, but also allowed him to insert whole blocks of information into the centre of his text. To do this the line numbers could no longer be stored as part of the text, but had to be computed by the editor when reading the source. This is a principle which has been followed by almost all text editors since. These early editors were developed as parts of primative filing systems, and were used in batch mode operation. The ammendments still had to be entered into the computer, either on punched cards, or on paper tape. It was common practice to hold the initial source of a large program on tape, and to ammend the card deck, or paper tape containing the editor commands manually, until it, too became unwieldy. A copy of the corrected source was then stored on tape, and the process was completed, until a satisfactory text was obtained.

It was realised at an early stage that the programmer was not really interested in the line numbers of the text which he wished to modify. He was, of course, interested in the text which the line contained. Also, by requiring that the complete line was retyped the editor commands included a lot of information which was not strictly necessary. Experiments were conducted into the use of context editors, using the text itself to identify the positions of the information to be altered during the mid-60's. It was now possible to change single characters, or groups of characters within a line without retyping the whole line. Two different types of editor were now developing, one contextual, and the other based on whole line operations. Users were unhappy and confused by the availability of two different editors, intended to perform exactly the same function. It was therefore decided to combine the functions of both editors, and so obtain the full benefit of both. Lines could be defined by absolute line number, or by its position relative to the current line, or by the characters which it contained. Character strings could be inserted, deleted or exchanged within the current line. By introducing the concept of a character pointer it was possible to handle lines containing repititions of the same string more easily.

All the editors discussed so far were intended to be used either in batch mode, or in peripherals, such as teletypes, which supported sequential input/output. With the coming of interactive computer systems, and visual display units which allowed the computer to display characters at any position on the screen it was possible to display a section of the file on the screen, and allow the user to identify the characters to be altered by moving a special character, called a cursor, to the desired position. Commands are required to move the text 'window' up and down the file, manipulate the cursor, and to perform the editing operation required. Editors were being used to manipulate program and date files on line in an ad hoc manner. Most editors available for use on microcomputers are of this type. They are intended for use under immediate control, from the keyboard. Commands are few, and are designed to allow the user to define the changes he requires with the minimum number of keystrokes. Usually the user issues one command and waits for the computer to inform him of the result before issuing the next. He can then carry on, or re-edit to correct the text, if his last command did not do what he wished. One useful facility which has been introduced into this type of editor is the ability to scan the whole, or part, of the text for a particular character string, and to edit every occurance of it. This allows you, for instance, to change the name of a variable identifier everywhere it occurs, very easily. Systematic spelling corrections can be dealt with in the same way.

More recently, with the rapid increase in the power of modern computers, very sophisticated editors have been used to search text files in a systematic way. This is almost always done in batch mode operation, with the programmer using the editor commands as, in effect, a special text handling language. The editor can be used to find error messages in a compilation listing, or to isolate the results of a large program which are of immediate interest, leaving the rest to be analysed at leisure. It is possible, using an editor, to locate, and print out the final results of a calculation, to set up a data file for analysis by another program, and to obtain the intermediate results on microfiche, for storage. To do this the editor must have powerful macro, looping and conditional programming in a traditional computer language. These facilities would be for too complicated for all but the most experienced programmer to use from the Keyboard. On large mainframes, however, it is more economical to implement all the required facilities in one large package, since those parts which are not used will remain on disk, and will not be loaded into the computer's main memory. This is not possible with most microcomputers, which do not have enough memory, or backing store to do this. It is therefore necessary to mount several different editors, each intended for use in a different area. The main editor will be required for the manipulation of programs and data before they can be processed by the microcomputer. Indeed some interactive computer languages, such as BASIC, include a very simple line editor as part of their specification. This is sufficient for most program editing needs, but is useless for handling data, and other text files. If a compiler, or assembler, is to be used a stand-alone assembler will also be required.

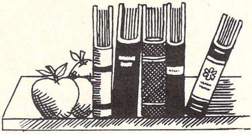
The copy for this article was produced on a popular microcomputer, using a very sophisticated text editor and formatting system, which provides many of the facilities available on commercial word processing systems costing many thousands of pounds. Although it is eminantly suitable for generating letters, and articles, it cannot be used to generate program and data files, which are to be input into assemblers, compilers and user programs since it automatically formats the text to fill the desired page size. Commands are included to centre lines, for headings etc., split the text into paragraphs, and to assist in the laying out of tables. Blocks of text can be moved around, as desired, so that the writer can re-organise his work, should this be necessary. All this makes the writer's task a lot easier.

Text editors are now far more sophisticated than in the earliest versions. The price to be paid is that, unlike in the early days, many programs designed to perform the same basic function, that is manipulating textual information, are required, each to be used in a different context. Most microcomputer owners will have several text editors, each designed to be used in a different context. Care must be taken to choose programs, which are similar in use, otherwise you will become confused when using the different editors. Each text editor should be selected to provide a different facility, and not to duplicate functions, which are already adequately covered by other programs. Typically an editor will be required to generate files to be read by any assemblers and compilers that you use, and a text editor/formatter to be used as a simple world processing system.

NOTE

The following article describes the development of the various editors used at one major site over a period of fourteen years, and will be of interest to anyone interested in seeing how a major item of software develops over such an extended period.

P. Hazel, 'The Development of Text Editing at Cambridge', IUCC Bulletin, 1, pp155-118 (1979).



SUPPORT MEMBERS OF THE COMPUTER RETAILERS ASSOCIATION . . .

CRA

COMPUTER RETAILERS ASSOCIATION

THEY WILL SUPPORT YOU.

For further details on the associations aims, membership, code of conduct etc.

Please contact: Mrs Gibbons,
Owles Hall, Butingford,
Hertfordshire, CE9 9PL.
Tel. (0763) 71209

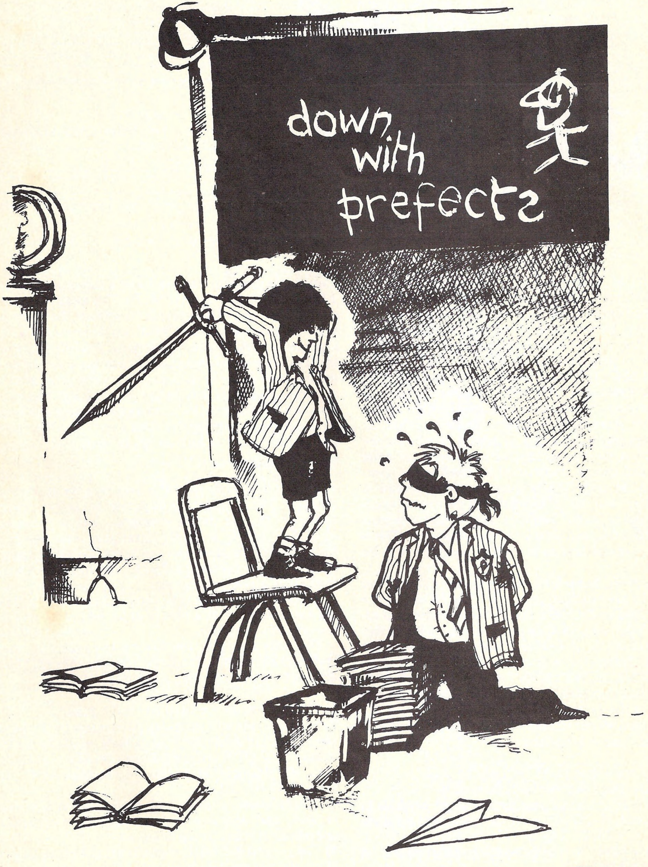
UNIVERSITY OF LIVERPOOL.
MICROPROCESSOR LABORATORY.
MICROCOMPUTER ASSEMBLER PROGRAMMING.
5 AFTERNOONS, STARTING TUESDAY 20TH MAY 1980.

This course gives a practical introduction to assembler level programming of microcomputer systems, particularly those based on the extremely popular 6502 processor. Practicals are arranged around the AIM-65 single-board computer, but the principles discussed can be applied with equal ease to the APPLE and PET, which are also based on the 6502.

For prices and further information, Please contact:

DR. M. D. Beer,
Computer Laboratory,
University of Liverpool,
P.O. Box 147,
LIVERPOOL.
L69 3BX.

Disassemble your monitor


down with prefects
Rigby

ALGOL68C ON THE Z80

Raymond Anderson

THIS article outlines an implementation of ALGOL68C on a Z80 microcomputer. The project is being carried out by the author during his degree course at Cambridge University to allow users of Z80 systems at Cambridge to write large programs quickly, and also to write large programs for his Nascom 1.

ALGOL68C [1] is a sublanguage of Algol 68 [2, 3] with some extensions. Algol68 is a powerful general purpose language based on the principles of Algol 60 but with greater power and flexibility. Readers unfamiliar with the language are urged to read a primer on the language, but most programs included here are quite readable and can be understood by reading the comments and the text. The language is well checked at compile time, and this, combined with the ease of accurate expression in the language means that run-time errors are less frequent than with most languages. This is of course a useful feature if reliable software is to be produced.

The Algol68C compiler is written in ALGOL68C and produces an intermediate code called Z-CODE [4] (see appendix 2). The Z-CODE must be interpreted or compiled into machine code for a particular machine. In this implementation the Z-CODE is compiled into machine code by a 'Translator' written in ALGOL68C.

ALGOL68C allows hashes (#'s) as comment symbols. Remarks enclosed by hashes are ignored by the compiler in the examples that follow.

The normal way of transferring ALGOL68C to a new type of machine is as follows:

1) Write a translator in ALGOL68C.
2) Translate the Compiler and Translator.
3) Write runtime library code for the target machine in machine code, Z-CODE or ALGOL68C and translate it
4) You now have a working compiler and translator on the new machine!

Unfortunately, the standard ALGOL68C compiler is a very large program which requires a minimum of about 120K bytes of store on an IBM370. This means that until a smaller version of the compiler is written, the compiler must be run on a larger 'host' machine—usually the Cambridge 'CAP' research computer. Initially therefore the translator is run on the same machine, and the size is not a critical factor in its design. The code generated by the cross-compilation is linked with a small library of routines for transput (I/C), storage allocation and common utilities (e.g. multiply). This means that it will run on machines with as little as 4K bytes of store.

The Translator

Z-CODE is a simple assembly language for an abstract machine. The compiler is told details of the instructions available on the target machine, and it produces Z-CODE which is easily translated into machine instructions. The translator reads the Z-CODE from a file, and produces output in INTEL hex format in what is essentially a one pass process.

Program segments may be compiled separately, allowing a user to prepare a library of useful procedures for use with his programs.

The main aim of the translator is to produce compact code without too much sacrifice of runtime speed. This is achieved by the following techniques:

1) Relative jumps are used wherever possible.
2) Standard subroutines taking parameters on the hardware stack are used where inline code would be bulky.
3) Some Z-CODE instructions are optimised to simpler ones. For example: HL: = HL * 4 becomes ADD HL, HL; ADD HL, HL etc.
4) The translator tries to keep track of what is in each register, and does not perform redundant or useless operations if it can avoid doing so.
5) The general Z-CODE registers are not firmly associated with real machine registers—allowing some operations to be done with the most suitable register, without continually swapping registers.

The translator does not at present allow the use of REAL or COMPLEX numbers, mainly in order to simplify the runtime system, and also because most of the work carried out on Z80's at Cambridge does not involve real numbers. It is possible to write routines to simulate floating point arithmetic to arbitrary precision in Algol68 of course.

The basic modes (types) available are:

INT	16 bit integer
CHAR	8 bit character
BOOL	8 bit boolean (TRUE OR FALSE)
BUTS	16 bit value—machine word.
STRING	0-255 characters

These can be built up into structures and arrays or 'united values', without any extra work being done by the translator. For example, a mode can be declared:

```
MODE PERSON = STRUCT (STRING first name,
second name,
INT      age,
BOOL  sex);
```

Then a PERSON can be declared (and initialised) thus:

PERSON subject := ('Anne', 'Stafford', 21, female);

Assuming 'female' is a boolean value such as TRUE (male being FALSE!).

Second name OF subject : = 'Smith' would be a valid operation in this instance.

The Translator program itself is about 5000 lines of well commented ALGOL68C which runs quite quickly. To apply optimisation, it reads the Z-CODE in 'Basic-Blocks'—sections of code in which there is no flow of control to or from other places. Each basic block is scanned to determine the ideal register usage, and then translated into machine code. Extensive error checking during translation means that faulty Z-CODE is detected, and this means users can safely write directly in Z-CODE if the fancy takes them.

Runtime System

Algol 68 allows recursion in procedures and dynamic declaration of array bounds. This complicates the runtime system compared with simpler languages.

Consider the program:

```
BEGIN print ('How much space?', newline);
          INT space; read (space, newline;
                    [O: space] INT array;
          # declaring a vector if integers #
          etc . . . .


END
```

The amount of space required for the array is not known until runtime, so it has to be allocated dynamically.

In addition to the above, there is a form of storage known as the HEAP. This allows objects to be generated inside a procedure (for example) and to remain in existence as long as some reference to them exists. This type of storage is useful in list processing, and is also used by STRINGS, which may change in length during their lifetime.

There are seven areas of store in Z80 implementation:

Program code—allocated before running (constant).

Read only data—combined with program code.

The Z80 stack—allocated about 40 bytes before the program starts.

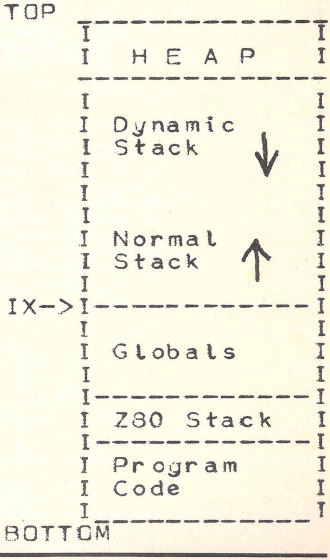
Global variables—Allocated places in store during translation.

Local variables—local to procedures which may be recursive. They are accessed relative to IX (the local stack) which is advanced when a procedure is entered.

Arrays (Global or Local)—A description of the array is kept on the Local or global stack, but elements are held on the 'Dynamic stack' so they may be released when needed no longer.

Heap Storage—is allocated by a routine 'heapgen' when required. It is in a separate area of store called the HEAP area.

Store Map:



The top of the dynamic stack is pointed to by a word on the local stack. When space for an array is required, the routine 'dyngen' is called to give the space required. This adjusts the pointer to the top of the dynamic stack and returns the address at which the array may be allocated.

If there is insufficient space, then a runtime error routine is called.

When a procedure is entered, a new set of variables is required, and also a record of the return address must be kept. To accomplish this, all variables local to a procedure are accessed as offsets from the IX register, which is used as a stack pointer.

The first two words on the local stack are used by the system. The first holds the address that the procedure was called from, and the second is used to chain back to the stack frame of the next enclosing textual level in order to access variables which are neither local to the current procedure or global.

Procedure Calls

Algol 68 procedures always return a result although sometimes the result is known as VOID, which means no value is really passed back. In this implementation, the parameters for the procedure are copied up the stack into locations where they will be accessible when the stack pointer moves forward on entering the routine. The result from the procedure call is returned in the HL register pair. In some cases the HL register will contain the address of a result if it cannot be held in 16 bits.

Entry to a normal procedure is as follows:

The space used by the current stack frame is added to the IX register, and the return address is saved at offset 0. If a chain value is provided in the DE register, it is stored at offset 2 on IX. The code of the procedure is now executed. At the end of the procedure, the return address is taken from offset 0, and the stack is retracted to its original state. In practice these actions are done by a routine call to ENTERS and a jump to LEAVE. The procedure word giving the increase in stacksize required. This word is accessible indirectly via the return address for use in retracting the stack, so the old value of the IX register need not be saved on routine call.

Example:

Consider the program:

```
PROC add two numbers = (INT a,b) INT: a + b;
INT x, y, z;   # Declare three integers #
x:=3;  y:=4; z:=add two numbers (x, y)
```

After assignment of 'x' and 'y', the stack will look like:

```
-----------------------------------------------------
__I_3_I_4_Iz=?I___I___I___I___I___I___I_->
```

Where each box represents two bytes. Just before the call, x and y are copied up as parameters:

```
-----------------------------------------------------
__I_3_I_4_Iz=?I___I___I_3_I_4_I___I___I___I_->
```

Now the procedure is 'called', setting IX to point to the base of the procedure local workspace. The return address is placed in offsets 2 & 3 as described above.

```
              IX->
-----------------------------------------------------
__I_3_I_4_Iz=?Iadr I___I_3_I_4_I___I___I___I_->
```

The routine now accesses its parameters as offsets from IX. The sum of the parameters is calculated and held in HL over the procedure return. The actual code produced is given in Appendix 1.

The stack after completing the routine call and assignment of 'z' will be:

```
-----------------------------------------------------
__I_3_I_4_I_7_I___I___I___I___I___I___I_->
```

Runtime Errors

EVEN the best programs occasionally encounter runtime situations that they either do not expect or cannot handle. A clear indication of what has gone wrong is useful, and if the place in the code where the error occurred can be indicated, the message is even more useful. Obviously the code has to be fairly compact so the amount of data available when an error happens is limited. In the Z80 implementation, it is possible to follow back the routines which are currently using the stack, and determine the start address of each one. When the code for a routine is output, the first seven characters of the routine name are generated before the body of the routine. This enables the error routine to print the names of the routines which have invoked the routine which cased the error. In addition, the parameters of the routines are the first few items on the stack, so the diagnostic message can print them out.

An example of a runtime error message is as follows:

```
++++ALGOL68C run time error: non digit detected in
'read int'
  stack: C832    heap: D6F0, dyn-stack: D220, top:D800
  routine        segment   args
  sysreadint     ZLIB      26 72 12 FF ED AD AA AA AA
  readint        BASE      26 72 29 64 AA 26 72 12 FF
  nexnum         MAIN      67 54 67 89 00 AA FD AA 21
  (main Line)    (no name)
```

This runtime error dump shows that 'nexnum' routine in the MAIN segment of the program called the 'readint' routine in the system 'BASE' library, which in turn called the system machine code routine 'systreadint' which detected a character which was not a digit while reading an integer from the input file. The hexadecimal numbers give the first 9 parameters on the local stack which may be useful of debugging. In this case they are not very helpful, but the error message is explicit.

Typical Uses

Many useful programs are available in ALGOL68C at

Cambridge. The disassembler used in this article will run out on Z80, although it needs quite a lot of store. A sequential logic simulator, a symbolic differentiator, a turing machine simulator, several sorting routines, a simple compiler for the 'A' language, and several games programs in ALGOL68C are also usable on the Z80.

Although programs written in ALGOL68C tend to take up more space than hand coded programs, the ease of programming in ALgol68 more than makes up for this in most cases.

The system described would be very useful for writing large commercial packages. There would be no source code to pirate, and no overhead of BASIC interpreter.

ALGOL68C allows the user to write CODE sections in most places a BEGIN—END block would be acceptable. These allow the user to write sections of machine code or Z-CODE in a program when special features are required. System programmers use such features extensively—indeed some the STRING operations are written in ALGOL68C but with CODE sections where the Z80 'block-move' instruction is appropriate. The operating system for the Cambridge CAP computer is written in ALGOL68C, and the language has proved suitable for the job.

Appendix I—Sample Programs

THE following programs are simple programs to show the type of code produced by the translator. The code produced by the translator has been disassembled by another program and comments have been added. Some of the calls to external routines will be to strange locations as these are usually assigned by the link editor.

```
# Example 1: The program in the text above #

PROC add two numbers = (INT a, b) INT: a + b;

INT x, y, z;

x :=3 ; y :=4; z := add two numbers (x, y)
```

```
; TITLE Output code for program
   ORG 00100H


START LD     HL,3              0100 21   03 00
                               0103 22   7A 00                 store 3 at x via HL
                               0106 11   04 00
      LD     ( x ),HL          0109 ED   53 7C                 00 store 4 at y via DE
      LD     DE, 4             010D 22   84 00                 store x at parameter 1
      LD     (y), DE           0110 ED   53 86                 00 store y at parameter 2
      LD     (L132),HL         0114 CD   27 01
      LD     (L134),DE         0117 80   00 increase in stack size
      CALL ADDT                0119 22   7E 00                 store result in z
      DW     L128              011C C3   00 00
      LD     ( z ),HL          011F 07      name of next routine
      JP     STOP              0127 CD   02 FF                 advance IX and save
      DB 7, 'addtwon'          012A DD   6E 04                   return addr.
ADDT  CALL ENTERS              012D DD   66 05                 HL:= parameter 1
      LD     L,(IX+004H)       0130 DD   4E 06
      LD     H,(IX+005H)       0133 DD   46 07                 BC:= parameter 2
      LD     C,(IX+006H)       0136 09        add them into HL
      LD     B,(IX+007H)       0137     and return to caller
      ADD    HL,BC
      JP     LEAVE
```

```
# Example 2: A recursive factorial program #

PROC factorial = (INT i) INT: IF i = 0
                              THEN 1
                              ELSE i*factorial (i-1)
                              FI;
INT a,b;
a: = factorial ( 2 );    b: = factorial ( a );
```

The code produced by this program is:

```
START   LD      HL,2            ; 21 02  00
        LD      (par1),HL       ; 22 82  00 store parameter for fac.
        CALL    FACT            ; CD 2C  01
        DW      L126            ; 7E 00  increase in stack size
        LD      ( a ),HL        ; 22 7A  00 put result at a
        LD      (par1),HL       ; 22 82  00 and pass it to fac.
        CALL    FACT            ; CD 2C  01
        DW      L126            ; 7E 00
        LD      ( b ),HL        ; 22 7C  00 put result at b
        LD      (par2),HL       ; 22 84  00 parameter for next call
        CALL    PRINTINT        ; CD 00  00 0000 filled by link editor
        DW      L128            ; 80 00
        JP      STOP            ; C3 00  00 0000 filled by link editor
        DB 07, "factori"        ; — name of routine.
FACT    CALL    ENTERS          ; CD 02  FF
        LD      L,(IX+004H)     ; DD 6E  04
        LD      H,(IX+005H)     ; DD 66  05 get i into HL
        LD      A,L             ; 7D
        OR      H               ; B4 see if 0
        JR NZ,L318              ; 20 05
        LD      HL,L1           ; 21 01  00 set HL to result
        JR      L348            ; 18 1E  and jump to end
L318    LD      L,(IX+004H)     ; DD 6E  04
        LD      H,(IX+005H)     ; DD 66  05 get i back
        DEC     HL              ; 2B subtract 1
        LD      (IX+00CH),L     ; DD 75  0C
        LD      (IX+00DH),H     ; DD 74  0D parameter for next call
        CALL    FACT            ; CD 2C  01
        DW      L8              ; 08 00
        PUSH    HL              ; E5 result of call
        LD      L,(IX+004H)  DD ; ~~EE~~ 6E  04
        LD      H,(IX+005H)     ; DD 66  05
        PUSH    HL              ; E5 push i
        CALL    MUL16           ; CD 0F  FF and multiply them
        POP     HL              ; E1 into HL
L384    JP      LEAVE           ;
```

Appendix 2—Examples of Z-CODE

IN case the reader is interested to see what Z-CODE looks like, here is the Z-CODE for the second example above:

```
B330 326 1 145 0 7 +172 *Z BASE*Z
F0 10 0 +2
F40 10 6 +202
K10
I255 6 0 P327 7 +176
R1 10 1
F40 10 6 +172
K10
F20 10 6 +172
F40 10 6 +202
K10
I255 6 0 P327 7 +176
R1 10 1
F40 10 6 +174
K10
F20 10 6 +174
F40 10 6 +204
K10
I255 6 0 P277 7 +200
T237 146 0 0
S331 factorial *Z
T246 327 331 0
F20 10 7 +4
F500 10 0 +0
K10
H711 0 P332
R0
F0 10 0 +1
H116 0 P333
L332
R0
F20 10 7 +4
F2 10 0 +1
F40 10 7 +14
K10
I255 6 0 P327 7 +10
R1 10 1
F24 10 7 +4
L333
R1 10 1
L334
R1 10 1
T247 327 331 0
U330 334
Z
```

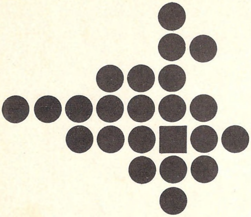
References

[1] S.R. Bourne et al. ALGOL68C reference manual, Cambridge University 1975

[2] van Wijngaarden et al. Revised Report on the Algorithmic Language Algol 68, Springer-Verlag 1976

[3] C.H. Lindsey and S.G. van der Meulen, Informal Introduction to Algol 68 (Revised), North-Holland 1977

[4] Bourne, Cheyney et al. Z-CODE—a simple machine, Cambridge University 1979



MICROCOMPUTERS & BIOCHEMISTRY

Dr R. J. Beynon Lecturer in Veterinary Biochemistry
University of Liverpool
P.O. Box 147
Liverpool L69 3BX

Introduction

AS a highly specialised subject, Biochemistry has been restricted in general to the Universities and Polytechnics, research institutes and large industrial laboratories. At all of these locations, large, mainframe computers are well established, and access to high-quality computing facilities has been available to most biochemists for many years. The number of biochemists who have made use of these facilities has been limited, in my opinion for several reasons.

Firstly, biochemistry is closer to being a descriptive than a mathematical science—we understand so little of the ways in which a vast array of complex molecules are assembled into a structure as complicated as a living human cell. (As a life scientist I take exception to Lauries' recent editorial in 'Practical Computing' [1]—when his idol—the 'wonder chip' produces a self replicating, adaptive, cognitive and independent machine in 70 kg (c.f. the human body); then he can make derisive references to 'butcher's shop materials'—until then, due respect to nature and her 3.0 billion years of research and development.) Consequently, models tend to be simple and are analysed readily without recourse to the application of mainframe facilities.

Secondly, the centralised mainframe with its aggressive operating system and slow turnround is a real deterrent to its use for small jobs—sometimes it's just not worth the walk to the nearest data centre.

Finally, biochemistry is an experimental science and the biochemist must be aware of the partition between acquisition and analysis of data—there is a danger of computers leading a 'gloss of sophistication' to a descriptive science.

And now, biochemists note the arrival of the microcomputer, replete with its entourage of attendant adjectives; 'approachable, affordable, local, personal . . . ' We have to ask ourselves whether these microcomputers offer anything new that mainframes couldn't deliver. What I hope to achieve in this article is to describe, using personal experiences to illustrate cases, the potential value that these machines possess in an experimental science. I doubt if I am particularly well qualified to provide much enlightenment and I shall not include any programs. I take full responsibility for my opinions and will be pleased to communicate with anyone who wishes to raise any points from this article.

An overview

There are three major areas where microcomputers could be used effectively in biochemistry:

(1) Teaching
(2) Analysis of 'hand input data'
(3) On-line analysis of output from analytical instruments. Trying to decide on a configuration that is suitable for all three areas is difficult, but a typical specification might include:

Hardware	**Software**
Disk drives	Accurate floating point routines
Hard copy—ideally graphic	Graphic routines
Easy interfacing to instruments	High level language
High-resolution graphics	Machine code enviroment

This specification is demanding and is probably met by few systems currently available. The configuration that we are using at present is shown in Fig 1, is based on an Apple II plus and cost in the region of £3,500 (Fig 1, Table 1). In addition, the biochemistry department at Liverpool has an 8k (shortly to become 32k) PET and printer.

Teaching applications

Noble thoughts of **teaching** the principles of, say, enzyme kinetics, through the use of a microcomputer are best forgotten, unless the department is willing to sit down for a few months and write first class software to run on the machines. Whilst I enjoy writing programs in BASIC, I am primarily a biochemist and make no aspirations to being a programmer. Even the expensive Chelsea Science Simulation program on enzyme kinetics is easily crashed, and I wouldn't like to present it to a naive user.

The potential for the use of microcomputers in biochemistry lies in the **illustration** of biochemical concepts. Demonstrations in tutorials, particularly with a decent size monitor and graphics, but under the control of the tutor, can make a valuable demonstration of an important principle. One fascinating area in which such a demonstration would be of considerable value lies in the illustration of the genetic code. (Forgive the brief tutorial that follows).

The basis of modern molecular biology resides in what is called 'the central dogma'. This states that the genetic material that is passed from one generation to the next—nucleic acid, **codes** for the major determinants of biochemical function—the proteins. Simply, the code is as follows: nucleic acid contains 4 **bases,** represented by the letters A, U, G and C. These 4 bases code for the 20 different 'building blocks' of proteins—the amino acids.

Consider the nature of this code; clearly it cannot be on the level of one base: one amino acid—because then the 4 bases could only code for 4 amino acids. Similarly, a pair of bases, such as AU, AG, CC etc could only code for 4^2 amino acids (= 16), and would not be sufficient for the twenty acids. In fact, triplets (or codons) of bases are used, giving $4^3 = 64$ possible codes for 20 amino acids. Since there are now more codes than amino acids the code becomes degenerate, with more than one codon specifying one type of amino acid. There are also two 'start of message' codons, AUG and GUG and three 'end of message' codons, UAA, UAG and UGA. Incidentally, each human cell contains approximately 800 million codons (equivalent to 600 megabytes of information) packed into a volume of 65 cubic microns—puts 64K RAM into shame doesn't it?

The process of taking a sequence of bases, looking for 'start' codons and deciphering the subsequent code until an 'end' codon is apparent is called 'translation'. Normally, this biochemical process is taught statically, but here is an area where a microcomputer would be of considerable value. We are preparing a program that allows the student to input a sequence of bases or use a computer generated random sequence and then translate it. The program could then alter one of the bases in the sequence and retranslate it (in biological terms this is a mutation) for example, mutation of UCA to UGA would cause premature termination of the process—UGA is an 'end' codon. Other work using computer generated sequences of bases have been described [2]. (We hope to have our program running for student use by October 1980).

Programs for the PET which simulate the experiments for sequencing of a chain of amino acids have been described [3] and the Chelsea Science Simulation project on enzyme kinetics is published by Edward Arnold [4]. For an introduction to the use of computers in biological education the reader is referred to the recent papers by Smythe and Lovatt [5] and Morgan [7].

On a more mundane level we shall provide statistical routines on the PET for use by students of Veterinary Biochemistry during their practical work—there are few better ways of teaching statistics than by the application of tests to the students' own data.

From an administrative aspect we are using the Apple II to maintain marks of veterinary science students throughout their two years of biochemistry. The use of a minicomputer to maintain a 'bank' of multiple choice examination questions has been described [6] but I suspect that storage limitations might impede such developments on microcomputers.

Analysis of hand input data and 'personal computing'.

There are a number of advantages associated with the acquisition of a microcomputer as part of the research equipment available to the biochemist. A series of programs that are designed within the requirements of a limited research group or which provide frequently-referenced information could remain in the machine (or on the disk in the drive) all day, and be accessed as needed. The software for this type of 'laboratory utility' program has to be elegant, however, as it will be used by many people, from the original author to students. Booting procedures and commands impede the user if he knows little about the machine or BASIC and are best circumvented. Many microcomputers with disk drives allow automatic loading and running of an application program immediately after power-up with no intervening steps. I consider that this is an essential element in building up sets of utility programs. Use of the **computer** then becomes very secondary to use of the **programs**—all the user needs to know is how to insert the disk and switch on the machine. If the software is of sufficiently high quality it should be impossible for the user to escape from the environment of the program and should never 'see' the interpreter or command level—the program should be crashproof.

The design of crashproof programs often seems to consume far more memory than the central mathematical routines. All input should be entered as strings—many tests are possible on string data, including scans for 'end of data' markers or 'help!' requests. More difficult are overflow errors, but with the BASIC interpreter available on the Apple II and several other machines the 'ONERR' function allows control of such errors. All that is needed is a jump to some routine that tells the user that an overflow problem is apparent before RESUMEing execution. We generally find that the mathematical routines are the simplest part of the program to write—embedding those routines into an error-proof program can take much longer. The following criteria are those that we try to apply when producing utility software for general use.

1. The program should resist attempts to crash if by the entry of mismatched or ridiculous data.
2. The program should resist 'control C' type attempts to half execution (I don't have a method for resisting

'RESET' on the Apple—any suggestions?)

3. The program should respond to a specified 'HELP!' key (e.g.: '?') and should give appropriate assistance according to the position in the program.
4. If the program goes away to perform long calculations it should inform the user. (Have you ever been asked to hold on during a telephone call and ended up replacing the handset because you don't know whether the silence at the other end means that you have been disconnected?)

I know that these are all obvious—but by writing them in—a 4k set of marks routines quickly becomes a 16k program. Some readers may be interested in the routine in Fig 2—this does some of what I feel is necessary—but does not include the ONERR overflow and control C checks. The subroutine would return with RF set to a value dictated by the keys pressed, and with valid ansers in TS$ and TV if RF = 1. A routine in the main program including an ON RF GOTO could then deal appropriately with the response of the user.

The types of programs that we are using or developing include routines for the design of buffer solutions, which maintain hydrogen ion concentrations at defined levels in experiments, statistical routines for group analysis or regression and routines for driving the plotter to produce fully annotated and drawn graphs for theses and for publication. Incidentally, I am against the common type of regression program that allows the user to fit a curve to data according to each of six relationships—you should decide which relationship your data conform to *before* performing any curve fitting.

Finally, within this context we use a database system on the Apple II to allow limited access to the scientific literature appropriate to our research. The program allows limited subject guided searching and gives a brief citation, a reference to a card index containing more information and a brief summary of the subject of the work. This is helpful in providing reading lists for new students or for running a rapid search through, say, the file containing the set of the 600 references of interest in 1975.

'On-line' analysis of data

Many manufacturers are now releasing sophisticated microprocessor controlled analytical instruments which boast an impressive repertoire of capabilities that may be user-defined. (One spectrophotometer has a locking key that disabled the control panel and prevents casual button pushers from altering instrument settings!) These machines are expensive (especially in the face of diminishing research funding) and are limited to the flexibility designed into the instrument. The alternative approach—to interface a microprocessor to an existing instrument—is attracitve. This is especially so with a microcomputer, as the availability of a high level language will allow the user to write his own analytical routines.

The majority of analytical instruments output information as an analogue signal—posing the first problem of interfacing to a microcomputer. The majority of microcomputers that are currently available use 8 bit micro-processors and so can handle data best in 8 bit chunks. The use of an analogue/digital converter that produces 8 bit digital data has limited application in analytical research, as the range of numbers representing the whole of the analogue signal can only be from 0 to 255—an effective resolution of about 0.5%. Again, one of the reasons that promoted us to acquire an Apple was the availability of 12 bit A/D converters—giving a resolution of 0.25%. Obviously, software is needed to take the 12 bit signal, stored in two consecutive locations in the scratchpad RAM for the converter, to a form represented in 8 bits—but in machine code this is apparently very rapid.

The application that we are planning is the interfacing of the Apple II to a spectrophotometer—an instrument that measures the absorption of light by different molecules. The spectrophotometer would normally be controlled by TTL level signals which are available on one of the ports of the Apple. The analogue signal from the spectrophotometer would be passed through the 12 bit A/D converter to the Apple which would initially do little more than store the data. Subsequent analysis, with trial output onto the monitor at a resolution of 0.6% or eventual output onto the plotter, could then be performed at leisure. (Fig 3). The software to drive the set-up would permit repeated, timed scans, subtraction of one scan from another, averaging of data, and would allow dumping of data to disc, output to the screen or to the plotter. All of this could be performed within the experimental limits defined at the outset (or brought in from disc) by the person currently using the machine.

One a more sophisticated level, differentiation of changing analogue signals or integration of areas under curves would be feasible. The major disadvantage of this application is that the microcomputer is commited to an instrument, precluding its use for other applications.

Conclusions

Without introducing too much biochemistry, I hope to have indicated the ways in which microcomputers can find application in teaching and research in this science. The ideas presented here are not new, but indicate what we are doing currently to make use of the new machines. Progress is invariably slow, and a few of the applications have progressed little beyond the paper stage. Perhaps in a cou,e of years time we will be able to claim to have integrated the microcomputer into our laboratory—demoting it from its somewhat exalted position to an important tool in teaching and research.

REFERENCES

[1.] *Laurie, P. (1980). Practical Computing, 3: editorial-para.*

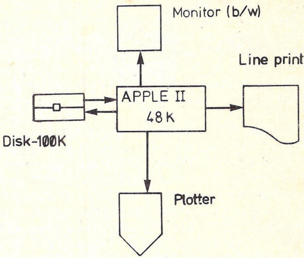
[2.] *Bryce, C.F.A. (1977). J. Biol. Ed., 11, 140-142.*
[3.] *Cunningham, P. (1979). Biochem. Ed., 7, 83.*
[4.] *ENZKIN–Edward Arnold Publishers Ltd., 25 Hill Street, London, W1X 8LL.*
[5.] *Smythe, R. and Lovatt, K. F. (1979). J. Biol. Ed., 13, 207-220.*
[6.] *Bryce, C.F.A. (1979). Biochem. Ed., 7, 17-18.*
[7.] *Morgan, M.R.J. (1979). Biochem. Ed., 7, 84-85.*

Table 1

An Apple II-based microcomputer system

Computer	: Apple II, 48k RAM, BASIC, 9 digit floating point routines, graphics.
Disk storage	: Single drive + controller 110 k/Disk, DOS 3.2
Hard copy	: Centronics 779 matrix printer + parallel interface
	Houston Instruments plotter + serial interface

FIG 1


Monitor (b/w)
Line printer
APPLE II
48 K
Disk-100K
Plotter

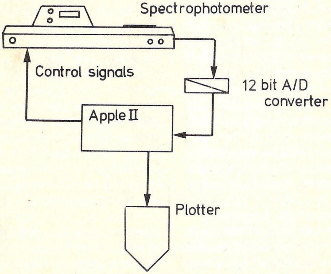
AN Apple-based microcomputer system in use at the Department of Biochemistry at Liverpool.

FIG 2

```
 1000 REM SUBROUTINE FOR NUMERIC INPUT
 1010 TS$="":TV=0::::::REM RESETVARIABLES
 1020 GETA$:IFA$=""THEN1020
 1030 IFASC(A$)=13THEN1140::::REM RETURN?
 1040 IFA$="?"ORA$="/"THEN1150:REM HELP!
 1050 IFA$="."ORA$="+"ORA$="-"THEN1160
 1060 :::::  FOR L=0TO9::REM NUMERIC?
 1070 :::::  IF ASC(A$)=48+LTHEN1160
 1080 :::::  NEXT L
 1090 GOTO1020
 1140 RF=1:GOTO1190
 1150 RF=2:GOTO1200
 1160 TS$=TS$+A$
 1165 PRINTA$;
 1170 GOTO1020
 1190 TV=VAL(TS$)
 1200 RETURN::REM RETURNS WITH RF,TV,TS$
READY.
```

AN inescapable numeric input routine. The subroutine returns with a numeric value in TV and RF = 1, or alternatively, if the 'help!' key was pressed RF = 2. Note that negative or real numbers could have been excluded by altering line 1050.

FIG 3


Spectrophotometer
Control signals
12 bit A/D converter
Apple II
Plotter

Use of a microcomputer for on-line analysis.

Sharp Machine Language Program SP-1002

Mike Shannon

Engineering Dept
Microdigital Ltd.

The Machine Language Program SP-1002 is the latest software from the Sharp stable for their MZ-80K Personal Computer. It allows the user to write, debug and save on tape machine code programs, to be used either by themselves, or in time critical parts of high level language programs.

The software comes in the form of a cassette tape which is loaded into the machine, and an instruction manual, which explains both the commands of the program and the instruction set of the Z80 microprocessor, which is the CPU in the Sharp MZ-80K.

The programs availability on tape, rather than in R.O.M. follows Sharp's policy with their BASIC Language tape. They claim it has the advantage of easier and cheaper updating of the software in the future.

Certainly the disadvantage of waiting for the tape to load in is not great. It takes just under 30 seconds for the tape to be read in, and the amount of free R.A.M. on board to be checked. The program lies from 1200 H through 2000 H and like the BASIC program the cold start is at 1200 H. On start up the program announces itself on screen and then proceeds to check the amount of R.A.M. present in the machine. In so doing it sets all this area to zero, thus destroying any data present.

Therefore if the Machine Code Program is to be re-executed, after jumping back to the monitor, say, then the warm start at 1260 H can be used. This does not perform a R.A.M. check and consequently any data in the free area from 2000 H upwards is safe.

Incidentally, since the Machine Language Program lies in the same area in memory as the BASIC program, the BASIC load command cannot be used to load this tape, loading must be done via the monitor or an overlay error will be detected. Once the tape is loaded we are ready to do some machine code programming. All the commands in this program use single character codes followed by hexadecimal arguments as appropriate.

A printer command implemented by typing a # character, enables anything printed on screen to be duplicated on an external printer (not available as yet). A second # character will disable the printer drive. Note that trying to use this command without a printer attached will give an error message and terminate the command.

Typing '!' as a command will return control to the monitor. When using this command take note of what was said earlier if a jump back to the Machine Code Program is to be performed.

The first command to use for actually placing your own program in memory is the Memory Write Command or W for short. After pressing W, the program inserts its own space and waits for a four digit hexadecimal address to start the writing operation. Any invalid characters (non hexadecimal) typed in for the address are not accepted and the 'bell' is sounded.

If the format of the arguments is not correct i.e. carriage return is pressed before typing the full argument, then several bells are sounded and the command is terminated, with the message 'INVALID' being displayed. If any of the arguments fall outside the free memory area, some question marks are printed and the command is again terminated.

These features, along with the auto spacing applies to most of the commands and takes a bit of getting used to, compared to 'dumb' executive programs on other machines. Once the start address has been specified, hexadecimal data may be typed in with the display showing 8 bytes to a line. The start address of each block of 8 bytes is shown at the left of the line. The command may be terminated by pressing Carriage Return.

One annoying feature is that the Delete Key is ignored. Thus if the wrong address is typed by mistake the command must be terminated and re-given. This problem does not occur with incorrectly typed data entries. This is because the 'cursor left' key can be used to step back up memory and make corrections. Another useful feature of this command is that if relative jumps are being used in a program, there is no need to work out the offsets yourself.

When the point has been reached where the offset must be typed, a full stop followed by the four digit actual address may be typed instead. The program will automatically calculate the relative offset in 2's compliment and insert it at the relevant point in the program.

Once your program has been typed in it may be checked before executing or saving to tape, by using the memory dump or M command. This requires two four digit arguments the start and finish addresses of the block to be looked at. The block is displayed in the same format as the W command, the start address and then a block of 8 bytes to a line. The dump may be broken at any time using the break key. On screen editing may be performed using the cursor control keys. The command is terminated with the CR key.

A useful related command is the transfer 'X' command which can be used for moving a block of data around in memory. It requires 3 arguments before it is executed, the start address of the originating block, the end address of the originating block and the start address of the destination block.

The next usual step in writing a machine code program is to save it on tape. This is a safety step. For, unlike programms written in high level languages like BASIC, if there is one incorrect byte in a M/C program the processor can go tearing off up memory destroying everything in its path, never to be seen again, (I speak from bitter experience) . . . At least with a copy on tape (albeit with a bug in it) there is a fighting chance of spoting the error by reloading the tape and examining a dump of the program on screen.

The tape save command 'S' asks for a filename for the program to be saved under. This may be any combination of characters up to a maximum of 16. (It's a pity other manufacturer's monitor programs don't have more than single character file names. Are you receiving me NASCOM?!!) The start and finish address of the block to be saved are then asked for.

After saving a program it may be verified using the V command. A file name may be specified to be searched for, or CR may be pressed and the verification done on the first program found on the tape. If any mismatches are found an error message is displayed.

A similar action is taken to the above when 'yanking' a program from tape at a later date using the 'Y' command. A filename may or may not be specified and any checksum errors are flagged on screen, otherwise the program is loaded into the same section of R.A.M. it was saved from originally.

Once the program has been safely saved on tape, the program may be executed. This is done using the 'G' command. This is the same as the GOTO $ command of the monitor and requires a four digit hexadecimal address at the start of the program.

If there are no bugs in the program every thing will work correctly, but if something is wrong, results will be very unpredictable. This is where the remaining commands come leaping to the rescue. Four commands are available to display the contents of the Z-80's on board registers. The 'A' command displays the Main register set (AF BC DE and HL) and allows modification by cursor control.

The 'C' command displays the complimentary register set (AF'BC'DE' and HL') in the same format. The 'P' command displays the Special Purpose registers PC SP IX IY and I and also allows modification.

Finally the 'R' command displays all the above three sets of registers but does not allow their contents to be modified.

The breakpoint or 'B' command allows you to set up to 9 breakpoints in your program with each being executed an individual number of times up to a maximum of 14. This is very useful for analysing conditional jumps out of loops. On typing 'B' all the current breakpoint addresses and execution counters are displayed. They may be modified or added to using the cursor controls.

Certain limits are put on where a breakpoint can be placed in a program. For instance, you cannot set a breakpoint at a DJNZ, an RST7 or a CALL type of instruction (i.e. one's that save the Program Counter on the stack. Applicable error messages are displayed if this is attempted or if more than 9 breakpoints are set. On executing your program by way of the 'G' command as soon as a breakpoint is encountered the 'R' command is automatically executed and all the CPU registers are displayed. Also program execution is halted. Execution may be advanced to the next breakpoint by typing 'G' if all proves OK in the first section of the program. By this powerful method bugs can usually be erradicated if breakpoints are set at relevant points in a program. Once a program is freed of bugs all breakpoints may be cleared by executing the clear breakpoint or '&' command.

One command missing from the package is a single step facility. This could have proved a valuable addition but since it is usually implemented using the NMI and some hardware, and the NMI on a MZ-80K is permanently pulled high, this could not be used. This command is not missed so much since the breakpoint facility is very comprehensive.

The manual that comes with the cassette tape has detailed descriptions of all the commands, along with other useful tables of information. Amongst the data is:-

A Memory Map of the MZ-80K, although details of the individual input/output addresses of the ports used are missing.

An explanation of how to link your machine—code routines into a BASIC program.

A list of the monitor and user callable monitor subroutines.

The Z-80 flags are described in detail and the rear half of the book includes all the Z-80 instructions listed three times. One list is by mnemonics in alphabetical order, one by hex-code in numerical order.

The first list will prove invaluable when hand assembling a program, and the second is a great boon when hand dissassembling someone else's program! All the instructions are also listed grouped by instruction type, with such details as numbers of clock cycles, no. of bytes and the way in which each flag is affected. This list is similar to that in the Mostek or Zilog Z80 CPU Technical Manuals.

The book is fine (after running through a Japanese—English cross compiler) for anyone like myself who is used to Z-80 machine code programming, but is not conversant with the MZ-80K, but for anyone starting out on low level programming I feel that additional guidance is needed. Either Nat Wadworths Z-80 Cookbook (Scelbi), Rodney Zaks Programming the Z-80 (Sybex) or William Bardens Z80 Microcomputer Handbook (Sams), (or all three!!!) would offer a good introduction.

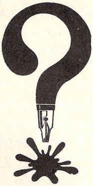
The one Assembly listing, included as a training program, which is well presented with flow charts etc, is a start, but brings back memories of trying to figure out how the 'Write the character set on the screen introduction to Z80 code' program supplied with Nascom 1 kits actually managed to work. Mind you at least Nascom gave you an assembly listing of the monitor R.O.M. and a technical manual on the PIO.

In conclusion the program offers some very powerful commands in assisting with the writing of machine code, but I do think a little more could have been included in the manual (at least a list of useful books) to help first time hex key-pad punchers. Well seasoned Z-80 freaks should have no trouble using the facilities provided and we can expect to see some fine programs sent in to the Gazette, if you get your fingers out!

Finally most of the commands are prevented from being used anywhere except in the free area. This makes saving a backup copy of the Machine Code Program (or taking a peek at it) impossible. However, the short program at the end of this article moves the main program into the free area, so that it may be interrogated by you at your leisure! Happy hand-dissassembling!

```
                    0010 ;
                    0020 ;  ZEAP ASSEMBLER RUN ON
                    0030 ; NASCOM 1. 29.2.80
                    0040 ;  LETS ****LEAP**** INTO ACTION
                    0050 ;
                    0060 ;  MIKE SHANAHAN
                    0070 ; MICRODIGITAL ENG. DEPT.
                    0080 ;  ******************************
                    0090 ; *     SHARP MACHINE LANGUAGE    *
                    0100 ; *     PACKAGE BACK-UP PROGRAM  *
                    0110 ; *******************************
                    0120 ;
                    0130 ;   THIS PROGRAM IS TYPED IN
                    0140 ;   USING THE MACHINE LANGUAGE
                    0150 ;   PROGRAM. AFTER EXECUTING FROM
                    0160 ;   $2000 THE M/C PROGRAM IS
                    0170 ;   MOVED UP IN MEMORY FROM
                    0180 ;   $2200 TO $2F00.
2000                0190            ORG   £2000
2000 210012         0200 START  LD    HL,£1200
2003 110022         0210 DEST   LD    DE,£2200
2006 01000D         0220 LENGTH LD    BC,£0D00
2009 EDB0           0230            LDIR
200B C36012         0240 END    JP    £1260
                    0250 ;
                    0260 ;  THE SECOND PROGRAM CAN BE TYPED IN
                    0270 ; NEXT,AT THE SAME LOCATIONS.THIS
                    0280 ;  IS SAVED ALONG WITH THE MOVED MAIN
                    0290 ; PROGRAM FROM $2000 TO $3000 BY
                    0300 ; USING THE 'S' COMMAND.WHEN THIS
                    0310 ; BLOCK IS RELOADED AT A LATER DATE
                    0320 ; USING THE MONITOR 'LOAD' COMMAND
                    0330 ; AND EXECUTED USING 'GOTO $2000',
                    0340 ;  THE MAIN PROGRAM WILL BE MOVED
                    0350 ; BACK TO $1200 TO $1F00 AND A COLD
                    0360 ; START WILL BE DONE AT $1200.
                    0370 ;
                    0380 ;  ******************************
                    0390 ; NB. DO NOT RE-EXECUTE AT $2000,
               0400 ; SINCE THE M/C PROG. NULLS ALL
               0410 ; FREE SPACE
               0420 ; ******************************
               0430 ;
               0440 ; THIS PROGRAM MUST ONLY BE USED
               0450 ; FOR MAKING A BACK UP COPY OF
               0460 ; AN ALREADY PURCHASED TAPE ***
2000           0470        ORG  £2000
2000 210022    0480 START1 LD   HL,£2200
2003 110012    0490 DEST1  LD   DE,£1200
2006 01000D    0500 LENG1  LD   BC,£0D00
2009 EDB0      0510        LDIR
200B C30012    0520        JP   £1200
               0530 ; M/C PROG. COLD START
```

* * *



Errata

to January 1980 edition

REVAS AND ZEAP

PAGE 20

Line 6 of the listing should read

8E00 0060 ORG£8E00 ; normal start for

PAGE 21

Line 4 of listing shouldread

8F3D 20.... 0330 DEFM/ORG£/

PAGE 23

Line 28 of second listing should read
OFA2 2A0C0C 0085 LDHL, (ARG1) ; first line requested

APPLE PIPS

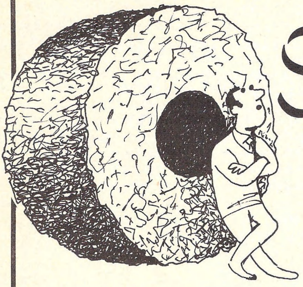
PAGE 48

Third paragraph (listing) lines 9 and 10 should read

40 DATA 230, 7, 202, 16, 244, 200, 192, 196, 144, 234,
50 DATA 96, 63

You, our readers, should notice that there were few errors in our second edition, in comparison to our first. We can only hope that the saying 'practice makes perfect' applies to our magazine and that with more experience in each issue you will find fewer mistakes.

Again, our apologies for these errors.



SUPERSORT

Roy Stringer
Sales Manager
Microdigital Ltd

MY first attempt at writing a numerical sort programme was typically awful. It seems that everyones first attempt at serious programming after giving up trying to write the best computer game yet, is to have a go at writing a programme of one sort or another 'OUCH'!!

I went straight to it without any researching into sort algorithms and my first attempt took about three hours to put 250 randomly generated numbers into correct ascending order (I've seen worse first attempts since).

As I was using an old R.O.M. Pet at the time, I was unable to get any sensible results for more than 250 numbers (although I would have had to sit up all night to take them anyway), due to the value replicating error in the old Pet R.O.M. However, it wasn't until I had made a few attempts that I found out about this fault.

If your not familiar with this problem and can get your hands on an old ROM Pet, then try opening an array of 500 variables and filling every other one with the numbers 1 to 250 in ascending order i.e.

```
10  DIMA (500)
20  FOR I = 1 TO 250  :  A(I * 2) = I   NEXT
30  FOR I = 1 TO 500  :  ?A(I)  :  NEXT
```

Then on listing them in line 30 you will find that some weird and wonderful things have happened to the array.

After overcoming that little problem I gave up my sort programme since it was 60 times slower than the only other sort programme I had seen running.

Then while I was on holiday over Christmas, I gradually realised (I thought) how this other programme must have been doing it, and resolved to write one of equal efficiency. Thanks to my not having read about sort algorithms before, I had found a very obvious but seemingly unusual algorithm.

'Of course' I thought, 'He's using the random number itself as an address for it's new position in the array.' And that, of course, meant that it had only to make one pass through the Jumbled Array.

This approach produced a very fast but very simple programme, the first version of which somehow found its way into last month's issue of this magazine.

This original version used a neat method of maintaining a record of each number's original position without having to use an extra array which would waste valuable memory. The usual method, to my knowledge, is to move each number's position record in a separate array in parallel with the movements of the number itself. Then when the completed array is listed, its sister 'position' array is listed as a record of where the number came from.

A dead give away as to how to get around this waste of memory is in line 100 of last months version. Firstly it prints the number in the original jumbled array pointed to by the current variable in the sorted array; A(B(I)). This is the next value in the ascending list. It then prints the current pointer in the sorted array which is of course the original position of the value; B(I). Finally it prints the new position; D.

This system received some criticism as it meant that the numbers themselves were not sorted although their addresses in the original array were. Further criticism was aimed at the fact that these pointers were packed quite loosely in an array three times the size of the original array. (See line 40). For this reason, it was necessary to vet out the zeroes scattered about in the pointer array when listing the sorted numbers.

I think that the importance of these features is purely a matter of opinion since just as many people seem to think that it is the end result that matters and not the way in which it is obtained.

Anyway, this month I have re-written the programme so that it can handle user values within any range that the machine can handle. (In this case an Apple). I have also removed the original position feature and instead, relocated the list of variables in ascending order in the original array, but I have also provided a list of modifications for putting the old features back. However, 57 seconds for sorting 1,000 numbers in a range of 0—1,000 is still pretty fast and the programme is still linear for a well distributed set of variables, i.e. 6 seconds for 100 numbers and 60 seconds for 1,000 numbers.

To run the programme you will firstly have to answer the question 'How many numbers do you want to sort?' with a value greater than zero.

Next you have to enter the lowest and highest allowable value. If those are entered as the same value then the highest value will be re-prompted. If they are the wrong way round then both values will be re-prompted. These values may be anything between -1E37 and 1E37 provided that the included range does not exceed 5E35. It uses these two values to determine the modifying constant needed to obtain an address from each value.

Having successfully got through that lot you will be asked 'Do you want to input your own numbers?'

I suggest that, if your response to the first question was a number greater than 50, and you are simply running the programme to test it, you should answer 'NO' to this question ('N' will suffice) and let the programme generate its own random numbers. If you reply 'YES' ('Y') then you will be prompted with a standard '?' for each value. Once you have entered the required number of values the sort will automatically begin.

The programme can be converted as mentioned above by changing the following lines:-

```
190  B(C) = I:NEXT
230  C = C + 1:PRINT A(B(I)), C
240  NEXT:END
300  IF A(B(C)) < = A(I) THEN 280
```

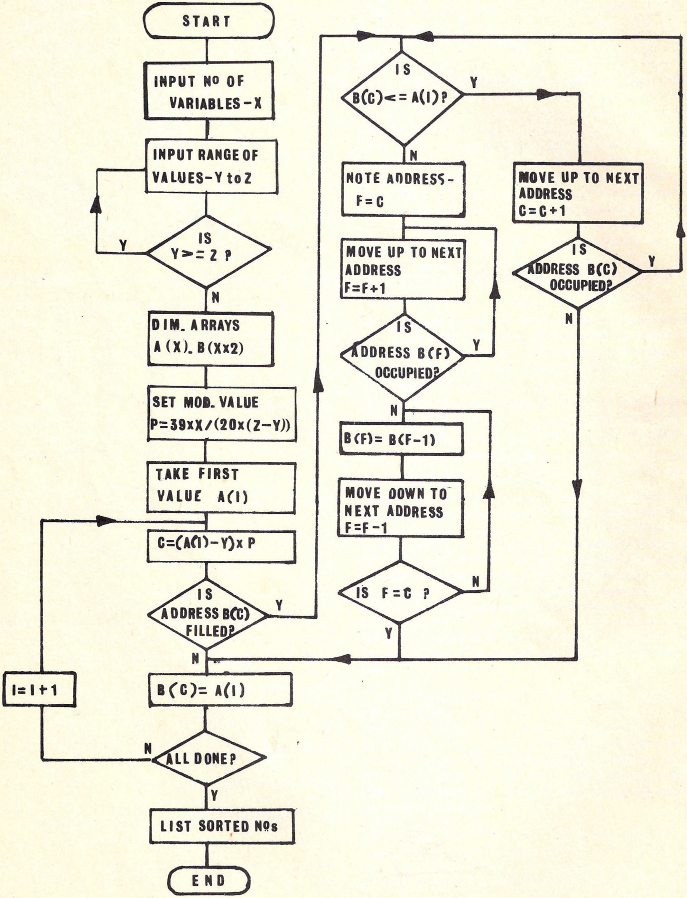
To conclude, I have compiled a list for both of the above version.

LINEARITY TEST	VERSION 1	VERSION 0
100 numbers from 0 to100	6	3
200 numbers from 0 to 200	12	6
300 numbers from 0 to 300	17	9
400 numbers from 0 to 400	22	13
500 numbers from 0 to 500	29	16
600 numbers from 0 to 600	34	19
700 numbers from 0 to 700	40	22
800 numbers from 0 to 800	46	25
900 numbers from 0 to 900	51	28
1000 numbers from 0 to 1000	57	31

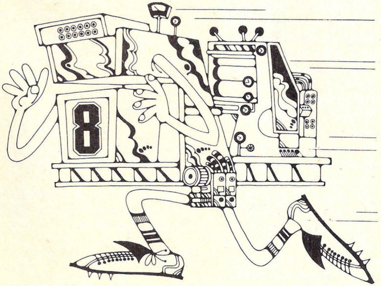
ODD TESTS

100 numbers from 0 to 1000	5	3
1000 numbers from 0 to -1E6 to 1E6	57	30
1000 numbers from 0 to 5E35	56	30
100 number from 1 to 1 + 1E-8	8	6
500 number from 1 to 1 + 1E-8	104	98
1000 numbers from 1 to 1 + 1E-8	423	400
100 numbers from 1 to 1+ 1E-7	5	3
500 numbers from 1 to 1 + 1E-7	35	24
1000 numbers from 1 to 1 + 1E-7	85	63

SUPER SORT


START
INPUT No OF VARIABLES – X
INPUT RANGE OF VALUES – Y to Z
IS Y >= Z ?
Y
N
DIM. ARRAYS A (X). B (Xx2)
SET MOD. VALUE P=39xX/(20x(Z–Y))
TAKE FIRST VALUE A(I)
C=(A(I)–Y)x P
IS ADDRESS B(C) FILLED?
Y
N
B(C)= A(I)
I=I+1
ALL DONE?
N
Y
LIST SORTED Nos
END
IS B(C)<= A(I)?
Y
N
NOTE ADDRESS – F=C
MOVE UP TO NEXT ADDRESS F=F+1
IS ADDRESS B(F) OCCUPIED?
Y
N
B(F)= B(F–1)
MOVE DOWN TO NEXT ADDRESS F=F –1
IS F=C ?
N
Y
MOVE UP TO NEXT ADDRESS C=C+1
IS ADDRESS B(C) OCCUPIED?
Y
N

```
]LIST
10  CALL  - 936
20  PRINT "HOW MANY NO'S DO YOU WANT TO SORT";
30  INPUT X: IF X < 1 THEN  VTAB 1: GOTO 20
40  PRINT
50  PRINT "RANGE FROM "; : INPUT Y
60  VTAB 1: PRINT  TAB( 7)"TO   "; : INPUT Z
70  IF Y > Z THEN  VTAB (3): GOTO 50
80  IF Z = Y THEN 60
90 R = Z - Y
100  DIM A(X): DIM B(X * 2)
110 P = ((39 * X) / (20 + R))
120  PRINT : PRINT "DO YOU WANT TO INPUT YOUR OWN NO'S"; : INPUT Q$: IF
     LEFT$ (Q$,1) = "N" THEN 160
130  REM  USERS NO'S
140  FOR I = 1 TO X: INPUT A(I): NEXT : GOTO 180
150  REM  RANDOM NO'S
160  FOR I = 1 TO X:A(I) = ( RND (8) * (Z - Y) + Y): PRINT I,A(I): NEXT
170  REM  ADDRESS NO'S TO SORTING ARRAY
180  FOR I = 1 TO X:C =  INT ((A(I) - Y) * P): IF B(C) <  > 0 THEN 300
190 B(C) = A(I): NEXT
200  REM  RETURN TO FIRST ARRAY
210 C = 0
220  FOR I = 0 TO X * 2: IF B(I) = 0 THEN 240
230 C = C + 1:A(C) = B(I)
240  NEXT
250  REM   LIST SORTED ARRAY
260  FOR I = 1 TO X: PRINT A(I),I: NEXT
270  END
280 C = C + 1
290  IF B(C) = 0 THEN 190
300  IF B(C) <  = A(I) THEN 280
310 F = C
320 F = F + 1: IF B(F) <  > 0 THEN 320
330 B(F) = B(F - 1):F = F - 1: IF F > C THEN 330
340  GOTO 190
```



SOFTWARE INTERFACE AND ACORN SYSTEMS

Laurence Hardwick

COMPUTER programs do not only have to interface with the outside world, in many cases they have to interface with other programs. This article describes interface methods and ways in which parts of programs may communicate with each other. Although particular reference is made to the 6502 processor and the interface specification of Acorn operating systems, the ideas should be of general interest to anyone writing programs whether for fun or for work.

FLAGS

Processor flags provide information about recently executed instructions and are usually tested to find the results of compare, load, bit, test or mathematical instructions. However the flags may also be used to provide information on what has happened in larger sections of code, for example at the end of a subroutine. This is done in a subroutine available in the Acorn System One monitor which fetches a key code from the keyboard. This routine returns with the carry flag set if a control key has pressed or clear for the key of a hex character, the code for the pressed key is also returned in the accumulator. Another example is given in Listing 1 which converts the ASCII code for a hexadecimal digit into its binary code and clears the carry flag or if a non hexadecimal digit is given to it the carry flag is set. The accumulator contains the ASCII code on entry and the binary code on exit from the routine.

Having called a subroutine which returns a result in the status register it is easy to branch dependant on the condition of the flags to take the appropriate action. However other operations may be called for before the conditional branch is made, in which case the state of the flag must be preserved. If the intermediate instructions do not affect the relevent flags all is well, otherwise the status must be saved, by pushing it onto the stack or by other suitable means, and restored before the branch instruction.

When setting or clearing a flag which is to be tested later it is worth considering the ways in which the flag's contents may be destroyed. The overflow flag in the 6502 is affected by only a very few instructions (PLP, CLV, RTI ADC, SBC) and hence once it has been cleared it is usually fairly easy to ensure that its status is not inadvertently altered.

When various tests are being performed it is possible to store the results in a byte in memory by using the rotate (ROR) instruction of the 6502. This instruction shifts the contents of the carry flag into the top bit of a memory location, and all bit in that location right by one bit.

A sequence of instructions something like:—

JSR TEST1 result C^1 in carry flag
ROR MEM
JSR TEST2 result C^2 in carry flag
ROR MEM
etc

will build up a byte in memory which contains the results in a sort of minature stack:—

C^3 bit 7 stack gets pushed
C^2 bit 6 down as each new
C^1 bit 5 result is added
" bit 4
"
"

The results can subsequently be rotated out of the memory location using the reverse instruction (ROL) and acted upon in order.

The use of flags is important in good programming, it is always useful to know which processor instruction affected what flags.

REGISTERS

The example program in Listing 1 demonstrates the transfer of data in the accumulator and the applications of this method are fairly obvious. The other processor registers may be used, index registers are not only useful as pointers and may be used to hold data in the same way.

MEMORY

When index registers are used as pointers they can point to a region of memory which contains data for a section of program or subroutine. An example of this is the routine to display messages on the Acorn display used in the mastermind program (see Issue 1) which is shown in Listing 2.

The messages are held in a table of display segment patterns to be sent to the display buffer by the routine shown. The X register is loaded with the lower byte of the address of the required message before the routine is called. Once the message is in the output buffer the scan display routine in the Acorn monitor can be used to show the message.

The use of the display buffer in the Acorn monitor is an example of data transfer by means of an agreed section of memory. In this case both the sending and receiving programs must know exactly what information will be stored in which memory locations.

PROGRAM MEMORY

It is possible to store fixed data for use by a subroutine in sections of program memory. This technique is particularly useful in the SC/MP program shown in listing 3.

The dats is stored in memory immediately following the subroutine call instruction (XPPC3). The subroutine scans through the data and it uses it as apprropirate until some escape data entry is encountered, at which time the pointer (P3) is pointing at the next instruction in the main program. A second XPPC3 then returns command to the main program.

DOCUMENTATION

Whatever data transfer techniques are adopted between program modules the format used should always be well documented. Not only will other people pick up your program and ask 'How do I give it data and where do the answers come out?', you will do the very same thing if you pick up an undocumented program two months after you write it.

Only a small number of idea have been mentioned in this article and different processors have different capabilities so some methods are more suitable for some machines than for others. It is useful to look at other peoples programs and see what sort of formats the use of software interface and to maintain a degree of flexibility in the input and output requirements of your programs.

ACORN SYSTEMS

To demonstrate some of the methods used on a 6502 system, and perhaps as some small step towards a standard, we can look at the following specification for the Acorn operating systems. These will support higher level languages in such a way that changes from a tape cassette to a floppy disk based system etc will not cause too large an upheaval, and provide a useful set of input/output facilities whatever the system hardware configuration.

LISTING 1

ASCHEX ACORN 6502 Assembler Page 01

```
0010: 0000          ASCHEX ORG $0000
0020:               **********************************
0030:               *   ACC contains ASCII on entry  *
0040:               *  ACC contains binary on exit   *
0050:               * on exit CC:- convertion done   *
0060:               *         CSET:- error           *
0070:               **********************************
0080: 0000 C9  30   ENTRY    CMPIM 'O
0090: 0002 90  0F             BCC  NONUM ASC11 code lower than 0
0100: 0004 C9  3A             CMPIM $ 3A
0110: 0006 90  0B             BCC  NUMOUT disit 0-9
0120: 0008 E9  07             SBCIM $ 07
0130: 000A 90  07             BCC  NONUM ASC11 code lower than A
0140: 000C C9  40             CMPIM $40
0150: 000E B0  03             BCS NONUM ASC11 code highert than F
0160: 0010 29  0F   NUMOUT    ANDIM $ F
0170: 0012 60                 RTS
0180: 0013 38       NONUM     SEC
0190: 0014 60                 RTS
ID
```

LISTING 2

MESOUT ACORN 6502 Assembler Page 01

```
0010: 0000          MESOUT 0RG  $0000
0020: 0000          TABLE *     $3000
0030: 0000          POINT *     $0020
0040:               ****************************************
0050:               *    MESSAGE OUTPUT TO DISPLAY         *
0060:               *ENTRY X:-LOWER BYTE OF MESSAGE        *
0070:               *                            ADDRESS   *
0080:               *  EXIT Y=$FF                          *
0090:               ****************************************
0100: 0000 A0 07    ENTRY LDYIM $07  8 digits to send
0110: 0002 B6 20          SIX POINT   X points to message on entry
0120: 0004 A9 30          LDAIM TABLE /load upper half of pointer
0130: 0006 B5 21          STA POINT  +01
0140: 000B B1 20    LOOP  LDAIY POINT load bit pattern
0150: 000A 99 10 00       STAAY $0010 store in output buffer
0160: 000D 88             DEY
0170: 000E 10 FB          BPL LOOP and loop 8 times
0180: 0010 60             RTS
ID
```

```
LISTING 3
PROGRAM

   .
   .
   .
LDH FREDH
XPAH 3
LDA FREDL
XPAL 3
   .
   .
   .
XPPC3    CALL FRED EXCHANGE P C + P3
   –   )
   –   )
   –   )  DATA TABLE
   –   )

NOP    )
   .   )  PROGRAM
   .   )  CONTINUES
   .   )

SUBROUTINE

FRED  LDA P3 @ 1 LOAD A AND INCREMENT P3
      EOR 08   08 IS CODE FOR NOP
      JNZ NORETURN
      XPPC 3   EXCHANGE P C + P3 TO RETURN
NORETURN EOR  08   RESTORE DATA
          .   )     PROCESS DATA
          .   )
          .   )
          .   )
          .
      JMP LOOP
```

O.S. SOFTWARE SPECIFICATION

THE 0S contains several routines which can be called to interface between user programs and the system hardware.

The routines are defined in such a way that they will be compatible with future acorn operating systems and higher level software, and are defined as follows:—

OSCL1 This subroutine interprets a string of characters held at 0100 terminated by a carriage return, as an operating system command. Detected errors are met with a brk. All processor registers are used, the decimal mode flag will be set to binary on exit.

OSWRCH This subroutine sends the byte in A down the output channel. This channel is usually treated as ASCII data and special action may be taken on ASCII control characters. In the COS the recognised control characters are the cursor movement and printer control characters.

No processor registers are destroyed.

OSCRLF This subroutine generated a line feed and then a carriage return using OSWRCH. A will contain 0D, N will be 0, Z will be 0 all other register will be as before.

OSECHO This subroutine fetches a byte from OSRDCH and then writes it out using OSWRCH. If a carriage return occurs in OSRDCH both a line feed and a carriage return are sent to as OSWRCH. A will contain the byte, N, Z and C are unknown, all other registers are unchanged.

OSRDCH This subroutine fetches a byte from the input channel into A. The state of N, Z and C is unknown, all other registers are unchanged.

OSLOAD This subroutine loads all of a file into a specified area of memory. On entry X must point to the following data in zero page:

INTERUPTS

The following action is taken on interupts.

```
NMI     PHA
        JMP  (NMIVEC)

IRQ/BRK
        STA  $FF
        PLA
        PHA
        AND  #$10     which interupt was it
        BNE  BRK
        LDA  $FF
        PHA
        JMP  (IRQVEC) it was an IRQ
BRK     LDA  $FF
        PLP
        PHP
        JMP  (BRKVEC) it was a BRK
```

RESET On reset the operating system is executed starting with the transfer of the vectors into page two.

The C0S uses locations $C0 upwards in zero page for scratch pad memory and these locations should be altered by user programs.

→ string of characters, terminated by 0D, which is the file name.

address in memory of the first byte of the destination.

bit 7 0 ignore above, use files address.

The data is copied by the operating system and is not harmed. All processor registers are used by the status is saved. A break will occur if the file cannot be found.

In interupt or dma driven systems a wait until completion should be performed if the carry flag was set on entry.

OSSAVE This subroutine saves all of an area of memory to a specified file. On entry X must point to the following data in zero page.

X→ → string of characters, terminated by 0D, which is the files name.

Where to put the data when reloaded.

address of machine code to go to if data is to be executed.

start address in memory of data

end address + 1 of the data.

The data is copied by the operating system and is not harmed. All processor registers are used but the status us saved.

In interrupt or dma driven operating systems a wait until completion should be performed if the carry flag was set on entry. A break should occur if no storage space large enough can be found.

OSBPUT This subroutine outputs the byte in the accumulator to a sequential write file, X and Y are saved, N, Z and C are unknown. In the COS interrupts are disabled during BPUT but the interrupt status is restored on exit. In other systems the files sequential byte pointer will be incremented after the byte has been saved.

OSBGET This subroutine returns the next byte from a sequential read file in A. X and Y are retained N, Z and C are unkown. In the COS interrupts are disabled during BGET but the interrupt status is restored on exit. In other systems the files sequential byte pointer will be incremented after the byte has been read.

On reset a set of vectors are moved into RAM in block zero which point to these routines. These vectors are in RAM so that they may be changed by a users program to point to other routines ie. serial interface ect. the vectors are as follows:—

0200	NMIVEC	NMI routine entry
0202	BRKVEC	BRK routine entry
0204	IRQVEC	IRQ routine entry
0206	COMVEC	operating system command line interpreter
0208	WRCVEC	write character to output subroutine
020A	RDCVEC	read character input subroutine
020C	LODVEC	load program subroutine
020F	SAVVEC	save program subroutine
0210	RDRVEC	ERROR
0212	STRVEC	ERROR
0214	BGTVEC	get byte from tape
0216	BPTVEC	put byte to tape
0218	FNDVEC	ERROR
021A	SHTVEC	ERROR

The vectors which point to error are there to allow for software expansion to sequential file handling. A call to error causes the COS to output.

Com?

a break is then executed.

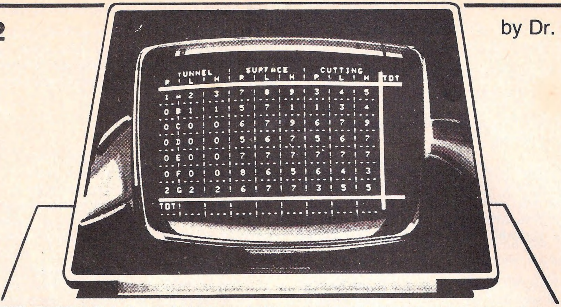
CALLS

As there is no jump to subroutine indirect to use these vectors the COS has the following calls in it.

```
FFCB   OSSHUT   JMP  (SHTVEC)    unused
FFCE   OSFIND   JMP  (FNDVEC)    unused
FFD1   OSBPUT   JMP  (BPTVEC)
FFD4   OSBGET   JMP  (BGTVEC)
FFD7   OSSTAR   JMP  (STRVEC)    unused
FFDA   OSRDAR   JMP  (RDRVEC)    unused
FFDD   OSSAVE   JMP  (SAVVEC)
FFED   OSLOAD   JMP  (LODVEC)
FFE3   OSRDCH   JMP  (RDCVEC)
FFE6   OSECHO   JSR  OSRDCH
FFE9   OSASCI   CMPIM#$0D
FFEB            BNE  OSWRCH
FFED   OSCRLF   LDA#$0A
FFEF            JSR  OSWRCH
FFF2            LDA#$0D
FFF4   OSWRCH   JMP  (WRCVEC)
FFF7   OSCLI    JMP  (COMVEC)
```

Part 2

by Dr. B. Allan



```
      TUNNEL    !  SURFACE  !  CUTTING
 R  !  L  !  H  ! R ! L ! H ! R ! L ! H   TOT
1 A 2 ! 3 ! 7 ! 8 ! 9 ! 3 ! 4 ! 5
---!---!---!---!---!---!---!---!---
0 B 1 ! 1 ! 5 ! 7 ! 1 ! 1 ! 3 ! 4
---!---!---!---!---!---!---!---!---
0 C 0 ! 0 ! 6 ! 7 ! 9 ! 6 ! 7 ! 9
---!---!---!---!---!---!---!---!---
0 D 0 ! 0 ! 5 ! 6 ! 7 ! 5 ! 6 ! 7
---!---!---!---!---!---!---!---!---
0 E 0 ! 0 ! 7 ! 7 ! 7 ! 7 ! 7 ! 7
---!---!---!---!---!---!---!---!---
0 F 0 ! 0 ! 8 ! 6 ! 5 ! 6 ! 4 ! 3
---!---!---!---!---!---!---!---!---
2 G 2 ! 2 ! 6 ! 7 ! 7 ! 3 ! 5 ! 5
TOT!   !   !   !   !   !   !   !
---!---!---!---!---!---!---!---!---
```

NUMERICAL ACCURACY OF MICROCOMPUTERS

INTRODUCTION

IN the first article in this series (Allan, 1980), I showed that microcomputers based on the 6502 processor using BASIC language interpreters could produce fairly accurate numerical results. In my conclusions to the article I noted that there seemed a possibility that the 'SQR' function in APPLESOFT (and PETSOFT) BASIC might not be as efficient as it should. I felt that this could be the case because the Cholesky inversion routine (also called the 'square root method') is sensitive to the efficiency of the implemented square root routine, and the APPLESOFT results were slightly worse than those of an IBM 7090 which had less bits for its fixed-point part.

I have checked this point and find the 'SQR' in APPLESOFT BASIC is fairly inaccurate. In the next section I give a user defined function 'FNS(X)' for finding a square root (accurate to machine limits); I then give some specimen examples of the relative accuracy of 'SQR(X)' and 'FNS(X)'. Following this I discuss the nature of the Cholesky routine, and reapply the Cholesky routine with 'FNS(X)' replacing 'SQR(X)' to the Hilbert Matrix examples of Allan (1980). A coding for the Cholesky routine and the Hilbert test program is given in the Appendix.

'FNS'—AN IMPROVED 'SQR'

NOBLE (1964:68) notes that '. . . it is worthwhile devoting a considerable amount of effort to developing an efficient square-root program . . . ' (by 'program' he means a machine-cooled function); unfortunately the square-root function in APPLESOFT BASIC seems to be sadly lacking in efficiency in efficiency and accuracy. The last statement can easily be justified by = recourse to Newton's method (Noble, 1964:26-30, esp Ex 2.2) and this method is the basis for the new function 'FNS'.

To get a feel for Newton's method try this program:

```
10 INPUT 'NUMBER';Z
20 GUESS = Z/2
30 FOR I = 1 TO 30
40 GUESS = 0.5 * (GUESS + Z/GUESS)
50 PRINT 'GUESS';I;'IS':GUESS
60 NEXT I
70 END
```

(note that this program is in [a basic a BASIC] as possible).

On running this program you can see just how quickly the successive guesses converge to the same number, the square root of the number you input. The key line is statement 40.

Normally we 'guess' the value of the square root in BASIC by use of the 'SQR' provided function. That is,

```
GUESS = SQR(Z)
```

and if 'S' denotes an improved estimate of the square root of 'Z', statement 40 would suggest

```
S = 0.5 * (GUESS + Z/UESS),
```

or

```
S = 0.5 * (SQR(Z) + Z/SQR(Z))
```

It is possible to continue by using 'S' in a further step in the iteration, but, if statement 20 is altered to

```
20 GUESS = SQR(Z)
```

and the program is then run, you will see how quickly successive guesses converge.

This can be utilized as

```
 5 H = 0.5
10 DEF FNS(X) = H*(SQR(X) + X/SQR(X))
```

where 'FNS(X)' gives the exact value of the square root. (If 'FNS(X)' does not give the exact value of the square root, then 'SQR(X)' must be very inaccurate. Try

```
 5 H = 0.5
10 DEF FNS(X) = H*(SQR(X) + X/SQR(X))
20 INPUT 'NUMBER';Z
30 G1 = H*(G1 + Z/G1)
40 FOR I =          1 to 30
50 G1 = H* (G1 + Z/G1)
60 NEXT I
70 G2 = SQR(Z)
```

```
  80 G3 = FNS(Z)
  90 PRINT 'TRUE ROOT IS';G1
 100 PRINT 'SQR IS';G2
 110 PRINT TAB(5);'RELATIVE ERROR IS';(G2-G1)/G1
 120 PRINT 'FNS IS';G3
 130 PRINT TAB(5);'RELATIVE ERROR IS':(G3-G1/G1)
```

and not that the relative error in 'FNS is always zero. (It is instructive to note that in SOBS BASIC on the ICL 1902T the relative error in 'SQR' is zero.)

TABLE 1 shows relative errors in 'SQR' (for APPLESOFT BASIC) over a wide range of values of Z: it can be seen that the errors (which vary from 7.20 E-10 to 3.37E-10) indicate constant inaccuracy in the 'SQR' function—apart from the four smallest calues of Z. The implications of these inaccuracies will now be considered for the case of the matrix inversion routine used in the first article (Allan, 1980), to see how errors in estimating elements are affected by the use of the improved square root function.

TABLE 1

RELATIVE ERRORS IN 'SQR' FUNCTION FOR APPLESOFT BASIC

Z	RELATIVE ERROR
Z	0
5	0
16	0
65	0
326	4.13E-10
1957	3.37E-10
13700	5.09E-10
109601	7.20E-10
986410	4.80E-10
9864101	6.07E-10

THE CHOLESKY DECOMPOSITION ROUTINES (2)

IF A is a symmetric positive semi-definite matrix (i.e. the determinant of A is greater than or equal to zero), then it is possible to find a real lower triangualr matrix L such that

$$A = LL\ T$$

where L^T is the transpose of L. L can be thought of as the matrix square of root of A, and in fact one needs to calculate square roots to obtain L. In the test program, of the Appendix, statements 470 to 640 calculate the lower triangular matric and stores it in the lower triangular portion of the matrix 'A(20,20)' including the diagonal. If the matrix 'A' is singular (determinant of zero or near to zero) then a 'pivot' 'A(J,J)' in line 510 will be less than a small amount 'E1' (set in line 200 to be equal to 1E-7), and so there is a jump around line 520 to lines 540 and 550—line 550 outputs a warning.

After the lower triangular matrix is placed in the lower triangular portion of 'A', subroutine 360 uses L to calculate the inverse matrix: successive columns of the identity matrix are placed in 'X' and subroutine 220 is called (the routine extends from 220 to 350). The corresponding column of the inverse matrix is returned in the same vector 'X' and stored in the appropriate column of 'B'. If one is performing a solution for a set of linear equations without needing the inverse matrix 'X' would contain the constant in the equation (e.g. in the example in the first paper the constants were X(1) = 199 and X(2) = 197), and one would use.

```
1000 GOSUB 470
1010 GOSUB 220
```

Note that in lines 520 we use the function 'FNS'

Statements 660 to 760 calculate the inverse Hilbert matrix of order N (= 2 to 10) which is then converted to estimate the Hilbert matrix. 'Q' in statement 870 calculates the exact value of the I'th J'th element of the Hilbert matrix, which is then compared to the estimate 'B(I,J)' with the relative error being found in 880. 'M1' holds the largest relative error.

This coding—suitably adapted, with multi-statement input per line—has worked successfully on 6502 processors, Z80 processors, as well as the ICL 1902T. Descriptions of the Cholesky method are referred in Allan (1980); a good, advanced, discussion is Ralston and Rabinowitz (1978: 410-437).

THE IMPACT OF 'FNS' ON THE CHOLESKY/HILBERT PROGRAM (3)

TABLE II shows results for the Cholesky/Hilbert program which are given for both the 'SQR' and the 'FNS' functions (partly taken from Allan (1980:TABLE I). The improvements in accuracy due to the use of 'FNS' are remarkable: the ratio between the relative sizes of errors is of the order of 1:100, and the 'FNS' program is very accurate up to the 7 x 7 matrix—and reasonably close for the 8 x 8 matrix, which could not be inverted by the 'SQR' program.

In the eariler paper it was noted that for an IBM 7090 the largest error was about 1.9E-4 for the 6 x 6 matrix—to be compared with the error of 5E-6 for the 6 x 6 matrix given by the 'FNS' program. This shows, I hope, that present day microcomputers are as accurate as many earlier macrocomputers **but one needs to be extremely careful in the coding of numerical procedures.**

TABLE II

LARGEST RELATIVE ERRORS IN HILBERT MATRICES, APPLESOFT BASIC

Order	Using SQR (1)	Using FNS
2 x 2	0	0
3 x 3	0	0
4 x 4	0	0
5 x 5	2.5E-5	0
6 x 6	5.8E-4	5.0E-6
7 x 7	1.6E-2	4.6E-4
8 x 8	*(2)	1.7E-2
9 x 9	*	2.3E-1
10 x 10	*	3.2E-1

Notes: (1) Taken from TABLE I of Allan (1980);
(2) Matrix is singular at this point according to this routine.

CONCLUSIONS

Worries about square roots do not only have an impact on multivariate procedures: in Lusty (1980) we are presented with an example of inaccurate square roots for the RM 380Z. Lusty shows how the square root of 50.000000* 50.000000 is calculated by the 380Z to be 49.999993 (a relative error of 1.4E-7). However, Lusty does not try to improve the accuracy of the square root (sq. root) **but notes that, if the true square root is integral,**

INT (SQR(Z) + 0.0001)

will equal the true square root. (He uses this information in a computer game 'Square Triangles—Solution'). Obviously if one does not understand the numerical basis of errors in functions such as 'SQR', then one will tend to use adhoc solutions to eradicate the errors. This is poor programming, and with complicated programs is also probably wrong. The final article in this series will discuss how functions can be improved, on the basis of numerical analysis and not of ad-hoc-ery.

FOOTNOTES

(1) **Proof:**
Let 'T' be the square root, and 'A' be the approximate square root. Define the relative error in 'A' by

$$e = \frac{(A - T)}{T}$$

thus

$$A = T(1 + e)$$

as 'T' is the true square root $Z = T^2$ and so

$$\begin{aligned} FNS(Z) &= 0.5*(A + Z/A) \\ &= 0.5*(A + T^2/A) \\ &= 0.5*(T(1 + e) + T^2/(T(1 + e))) \\ &= 0\ 5*(T(1 + e) + T/(1 + e)) \end{aligned}$$

and thus if e is very small $(1 + e)-1 = 1-e$ and so

$$\begin{aligned} FNS(Z) &= 0.5*(T(1 + e) + T/(1 + e) \\ &= 0.5*(T(1 + e) + T(1 - e) \\ &= 0.5*(2*T) \\ &= T \end{aligned}$$

QED

(2) Technical, can be safely ignored

(3) Non-technical, do not ignore

(4) 0.5*(49.999993 + 2500/49.999993) = 0.5*(49.999993 + 50.000007) = 0.5*(100) = 50

REFERENCES

Allan G.J.B.
1980 The numerical accuracy of microcomputers (1): Macrocomputer BASIC compared to mircrocomputer BASIC. **Liverpool Software Gazette,** (2)

Lusty T.
1980 Problem page. **Computing Today,** 1(11).

Noble B.
1964 **Numerical Methods: 1 Iteration, Programming and Algebraic Equations.** Edinburgh: Oliver and Boyd.

Ratton A. and Rabinowitz P.
1978 **A First Course in Numerical Analysis** New York: McGraw-Hill

APPENDIX

```
 10 REM
 20 REM****************************************
 30 REM
 40 REM       HILBERT MATRIX TEST PROGRAM
 50 REM
 60 REM       AUTHOR—G J BORIS ALLAN
 70 REM
 80 REM****************************************
 90 REM
100 LET H=0.5
110 DEF FNS(X)=H*(SQR(X)+X/SQR(X))
120 DIM A(20, 20), B(20, 20), X(20)
130 PRINT 'MAX ORDER ?';
140 INPUT N1
150 FOR I=0 TO N1
160 LET X(I)=0                         Data Definition Section
170 FOR J=0 TO N1
180 LET A(I, J)=0
190 NEXT J, I
200 LET E1=0. 1E-6
210 GOTO 650


220 FOR I=1 TO N
230 FOR J=0 TO I-1
240 LET X(I)=X(I)-X(J)*A(J, I)
250 NEXT J
260 LET X(I)=X(I)*A(I, I)
270 NEXT I                                Back Subtitution
280 LET X(N)=X(N)*A(N, N)                 for array X
290 FOR I=N-1 TO 1 STEP -1
300 FOR J=1 + 1 TO N
310 LET X(I)=X(I)-A(I, J)*X(J)
320 NEXT J
330 LET X(I)=X(I)*A(I, I)
340 NEXT I
350 RETURN


360 FOR I1=1 TO N
370 FOR J1=1 TO N    X is If' Column of Indentity
380 LET X(J1)=0      Matrix
390 NEXT J1
400 LET X(I1)=1
410 GOSUB 220        Back  Subtitution
420 FOR J1=1 TO N
430 LET B(I1, J1)=X(J1)
440 NEXT J1               B (Il,) is Il' th Column of
450 NEXT I1               Inverse Matrix
460 RETURN
```

```
470 FOR J=1 TO N
480 FOR K=0 TO J-1
490 LET A(J, J)=A(J, J)-A(K, J)*A(K, J)
500 NEXT K
510 IF A(J, J)<=E1 THEN 540
520 LET A(J, J)=1/FNS(A(J, J))
530 GOTO 560
540 LET A(J, J)=0
550 PRINT '#######';J
560 IF J=N THEN 630
570 FOR I=J+1 TO N
580 FOR K=0 TO J-1
590 LET A(J, I)=A(J, I)-A(K, I)*A(K, J)
600 NEXT K
610 LET A(J, I)=A(J, I)*A(J, J)
620 NEXT I
630 NEXT J
640 RETURN
```

Forward Cholesky Elimination for Matrix A

```
650 FOR N=2 TO N1
660 LET P=N
670 FOR I=1 TO N
680 IF I=1 THEN 700
690 LET P=((N-I+1)*P*(N+I-1))/((I-1)*(I-1))
700 LET R=P*P
710 LET A(I, I)=R/(2*I-1)
720 FOR J=I+1 TO N
730 LET R=-((N-J+1)*R*(N+J-1))/((J-1)*(J-1))
740 LET A(I, J)=R/(I+J-1)
750 NEXT J
760 NEXT I
770 PRINT
780 PRINT
790 PRINT 'HILBERT MATRIX OF ORDER';N
800 PRINT
810 GOSUB 470
820 GOSUB 360
830 PRINT
840 LET M1=0
850 FOR I=1 TO N
860 FOR J=I TO N
870 LET Q=1/(I+J-1)
880 LET Q=ABS(Q-B(I, J))/Q
890 IF Q<M1=Q
900 LET M1=Q
910 NEXT J, I
920 PRINT 'MAX REL ERROR IS'; M1
930 PRINT
940 PRINT
950 NEXT N
960 END
```

Main Program

MICRODIGITAL BOOKS

BEST SELECTION-BEST PRICES-BEST SERVICE
25 Brunswick Street, Liverpool 2. Tel: 051-236 0707 (Mail Order)
051-227 2535 (All Other Depts)

Accounts Payable & Receivable	£9.95
Active filter Cookbook	£10.90
Adaptize Info, Processing	£8.75
Advanced Basic	£4.00
Algorithms & Data Structure Equals Programs	£14.00
Anatomy of a Compiler	£15.00
Apl. — An Interactive Approach	£9.50
Artificial Intelligence	£12.00
Artist & Computer	£3.96
Art Computer Programming Vol. 1.	£9.50 S/B £16.45 H/B
Art Computer Programming Vol. 2.	£16.45
Art Computer Programming Vol. 3.	£16.45
Assembley Level Programming for small computers	£12.76
Analysis and design of Digital Circuits and Computer Systems	£16.40
Accent on Basic	£4.95
About Computers	£6.95
Active Filters	£6.45
Analog/Digital Experiments	£7.15
A guided tour of Computer Programming in Basic.	£4.16
A quick look at Basic	£4.45
Aple II Operators Manual	£5.50
Apple II Integer Basic manual	£3.55
Apple II Applesoft Extended Basic Manual	£5.75
Advanced Business, Billing, Inventory, Invstments, Payroll	£26.95
An Introduction to Your new Pet.	£1.00
Basex	£5.60
Basic A Hands on Method	£6.50
Basic and the Personal Computer	£10.00
Basic, Basic.	£4.80
Basic Computer Games Micro.	£5.50
Basic from the Ground Up	£7.00
Basic Microprocessors and the 6800.	£7.20
Basic with Business Application	£8.40
Basic with style/Programming proverbs.	£3.60
Basic WorkBook	£3.70
Best of Creative Computing Vol. 1.	£6.95
Best of Creative Computing Vol. II.	£6.95
Byte Book of Computer Music	£7.00
Beginners Glossary and Guide	£5.75
Beginners Basic	£7.50
Byte Vol I	£8.45
Basic — a unit for Secondary Schools	£4.45
Basic Programming	£6.95
Basic Primer	£6.95
Collection of Programming Problems and Techniques	£12.00
Computer Crime	£12.00
Computer Data Directory	£3.98
Computer Lib/Dream machines	£5.95
Computer Models of Thought and Language	£17.00
Computer Power and Human reasons	£4.76
Computer resourse book Algebra	£4.00
Computer Science a First Course	£15.00
Computer Science Programming in Fortran IV	£7.80
Computer/Science Projects and Study problems	£8.75
Concurrent Pascal Compiler	£6.40
Conference Proceedings of the 1st West Coast Computer Fair	£9.56
Conference Proceedings of the 2nd West Coast Computer Fair	£9.56
Conference Proceedings of the 3rd West Coast Computer Fair	£9.58
Consumer Guide-Personal Computers & Micros.	£5.00
Content Addressable Parallel Processors.	£11.20
Computer Dictionary (sams)	£6.95
Computer Dictionary & Handbook	£11.95
Computer & Program Guide for Engineers.	£9.45
Computers for the Physicians Office	£15.50
Computerisation for small Business	£9.95
Computer Quiz Book	£5.45
Computer Programs that Work.	£2.95
Computer Music	£7.00
Computer Data Directory	£3.98
Computer Rage (a board game)	£6.95
Clarcia's Circuit Cellar	£5.60
Designing with TIL Circuits	£24.80
Design of Well Structured Prog's	£10.00
Dictionary of Microcomputing.	£10.00
Digital Computer Fundamentals.	£14.25
Dr. Dobbs Jounral Vol. 1.	£10.00
Digital IC Equivalents & Pin Connections.	£1.25
Designing M/Computer Systems	£5.40
Editor/Assembler Systems for 8080/8085 Based Systems	£11.96
8080 Programmer Pocket guide	£1.95
8080A Bugbook	£7.65
8080 Programming for Logic Design	£5.95
8080A/8085 Assembler Language Programming.	£6.95
8080/8085 Software Design	£7.65
8080 Machine Language Programming for Beginners	£5.10
8080 Microcomputer Experiments	£10.25
8080 Standard Monitor	£9.95
8080 Standard Editor	£9.95
8080 Standard Assembler	£9.95
57 Practical Programs Basic	£6.40
First Book of Kim	£7.00
Fortran Coloring Book	£5.56
Fundamentals of Data Structure	£15.00
Fundamentals of Computer Algorithms	£15.00
Fundamentals and Application of Digital Logic	£6.00
From the Counter — Bottom Line	£8.75
Fundamentals of Digital Comps	£7.25
Fortran Programming	£6.75
Fortran Workbook	£4.75
Fortran Fundamentals	£3.45
Fun with Computers & Basic	£5.45
555 timer	£5.35
50 Circuits using 7400 Series IC's	75p
Fortran Fundamentals a short course	£2.95
Game Playing with Basic	£5.50
Game Playing with Computers	£10.20
General Ledger	£9.95
Getting Involved with Your Own Computer	£4.75
Getting Acquainted with Micros	£7.95
Guide to Scamp Programming	£4.00
Games with a Pocket Calculator	£1.75
Games, Tricks & Puzzles for a Hand Calculator	£2.49
How to Build A Computer Controlled Robot.	£5.95
How To Profit from Your Personal Computer	£5.50
How To Program Microcomputers	£7.50
How You Can Learn To Live With Computers	£7.00
How to buy/use Mini-Micros	£7.50
How to Build a Working Digital Computer	£4.60
How To Package Your Software	£27.50
Home computer Revolution	£2.75
Home Computers: A Beginners Glossary and Guide.	£4.95
Hobby Computers Are Here	£3.95
Incredible Secret Money Machine.	£4.75
Interface Circuits Data Book	£3.50
Intro to Artificial Intelligence	£14.80
Intro to Computer Prog	£4.75
INTRODUCTION TO MICROCOMPUTERS.	
Vol 0	£5.95
Vol II	£18.95
Vol I	£5.95
Vol III	£11.95
Intro to Personal & Business Computing	£4.95
Intro to Prog. Prob. Solving with Pascal. Schneidser Et A1	£8.50
Instant Basic	£7.20
Introduction to Basic	£7.15
Interfacing and Communications	£4.95
Introductory Experiments in Digital Electronics Vol I	£8.75
Introductory Experiments in Digital Electronics Vol 2	£7.75
Introduction to TRS — 80 Graphics	£5.25
Instructor's manual for Fortran Programming	£6.95
Illustrating Basic	£2.25
IC Optamp Cookbook	£8.95
Introduction to Microprocessors & Computing	£2.40
Kim 1 Users Manual	£5.00
Linear Control Circuits Data Book	£4.00
Link 68 — M6800 Linking Loader	£5.50
Little Book of Basic Style	£4.75
Linear IC Principles Experiments	£7.16
Math Elements for Computer Graphics	£10.40
Microcomputer — Based Design	£8.00
Microcomputer Handbook	£15.96
Microcomputer Primer	£6.35
Microelectronics	£3.25
Micro Problems Solving Using Pascal	£7.50
Microprocessor Interfacing techniques	£7.95
Microprocessor Lexicon	£2.00
Microprocessor System Design	£14.00
Microprocessor From Chips to Systems	£7.50
Mind Appliance	£4.75
Modern Operational circuit Design	£18.60
Mondeb M6800 Monitor/debug	£3.50
Microcomputers at A Glance	£6.45
Micro/Computer Interface with the 8255 PPI Chip	£10.00
Microformula (TM)	£2.95
Microsoft (TM) Basic	£7.00
Microprocessor Basics	£7.60
Micros — New Directions for Designers	£7.60
Microprocessors Data Manual	£5.50
Micros for Business Applications	£7.15
More Basic Computer games	£5.50
Microcomputer Potpourri	£1.75
Modern Operational Circuit Design	£18.60
Microprocessor Basics	£7.60
Microprogrammed Apl Implementation	£14.75
Mos Memory Data Book	£2.80
Microprocessor Encyclopædia Vol 2.	£7.45
9900 Family systems Design & Data Book	£7.95
1975 U.S. Comp Chess	£4.75
1976 U.S. Comp Chess	£4.75
NCR Basic Electronics	£7.15
NCR Data Communications	£7.15
NCR Data Processing	£6.35
Nascom 1 — Hardware Notes	£1.50
Nascom 1 — Seminar Notes	£1.50
Optoelectronics Data Book	£3.50
Osbourne Updated Subscriptions	
Vol. 2. (6 issues)	£18.95
Vol. 3. (6 issues)	£18.95
Vol. 2 & 3. combined	£30.00
Binders 2 or 3	£5.75
Pascal User Manual & Report	£5.25
Payroll with cost accounting	£12.00
Power Semi-conductor Data Book	£7.50
The Mighty Micro	£5.50
Z — 80 Programming Manual	£4.50
TTL Databook	£7.00
TIS Workbooks (1-6) each	£3.00
TI S 6800 Tracer	£4.80
TTL Cookbook	£6.95
Peanut & Jelly Guide to Computing	£6.45
Tne Best of Micro	£5.50
Z — 80 and 8080 Assembly Prog.	£4.80
6500 Programming manual	£10.50
6500 Hardware Manual	£10.50
Software Gazette	50p

WELCOME
Prices include Postage and Packing anywhere in the UK.

ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC

ETC ETCETERA ETC

ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC

THE COMPUMAX

By Dr. Andrew M. Veronis

MICROCOMPUTERS have become as abundant as digital wristwatches and calculators. When a new microcomputer is introduced into the market, the people now say, and rightly so, 'Oh, another one of those!' What the latter statement really means is that 'unless this new microcomputer has much more to offer, we don't want it.' A statement which makes it tougher for the designer to dream up new revolutionary circuits.

The COMPUMAX is indeed a new microcomputer. Furthermore, the COMPUMAX is indeed built around new and revolutionary circuitry. The description of the circuit and a discussion on the computer's software capabilities will undoubtedly convince even the greatest unbelievers.

The new microcomputer uses the ever so popular Motorola MC6809 CPU as its main brain. Actually, computerists who are not up to par with the 6809 as yet have a choice of CPU's. The main board provides sockets for either the 6809 or the 6808 (the latter being an offspring of the MC6802).

The microcomputer offers full colorgraphics (8 colors), both at low resolution (32 x 64) and at high resolution (250 x 190). The MC 6847 is used for this purpose, and a PAL conversion circuit has been designed to fulfill the needs of European enthusiasts. Colorgraphics are generated via a very powerful BASIC language which is described in the software discussion of this article.

The COMPUMAX is equipped, as standard, with a super fast cassette system (speeds selectable from 1200 baud to 9600 baud). The cassette system is capable of driving two recorders, and has motor control capability. A full description of its software capabilities is again given in the software discussion.

Additional features of this revolutionary microcomputer which came as standard are the following:

16 Dynamic RAM , expandable on board—to 48 K(sockets and decoding are already provided).
Floppy disk controller circuit (capability up to four mini drives).
Parallel interface.
Serial Interface.
RS-232/20 ma interface.
2 user PIAs.
1 user ACIA.
10K Extended BASIC in ROM.
4K operating system monitor in ROM(including cassette and disk monitors).

However, enough said (not hardly!) about the hardware. This is a software oriented publication. So, let us march onto the software.

THE CASSETTE SYSTEM

The Compumax cassette system is truly unlike any others. It provides high performance storage for the computer. The major design goals for the system were speed and data reliability. In addition to reading and writing in high speed binary, the system is also capable of reading cassettes in the Kansas City Standard S1-S9 format without any modifications.

Here are some of its software capabilities:

SAVE: Preserves a named file on cassette from memory locations.
DIRECTORY: Lists the identification segment of a file on cassette to the system terminal. If the optional argument (FILENAM) is used, the tape head is positioned precisely at the End-Of-File position.
LINK: Preserves a named file on cassette in the same manner as the SAVE command except that a linkage is created to the next file on the cassette.
LOAD: Reads and loads a file from cassette.
RUN: Reads and stores a file from the cassette and transfers to the address specified by (TRANSFER) in the save command.
GO: Executes a memory resident program, either a previously loaded file or a program which was entered into memory through the terminal.
VERIFY: Reads a file from the cassette and compares each byte of the file with the corresponding byte in

memory.

MOVE: Copies the absolute binary contents of a block of memory specified by (FROM BEGIN) through (FROM END) into a memory block starting at (TO BEGIN).

COMPARISON: Performs a byte-by-byte comparison of the absolute binary contents of a block of memory with the contents of a memory block.

FIND: Searches memory in the range of (BEGIN) through (END) for all occurrences of (BYTE) or (WORD) and prints the corresponding addresses on the system terminal.

EXIT: Returns control to the Operating System Monitor.

ON: Turns on the cassette drive motor to allow manual operations (rewind, etc).

DRIVE: The prefix Drive is an integer 1, or 2 which specifies the cassette drive on which the command is to be executed.

THE BASIC INTERPRETER

One of the fastest BASIC interpreters in existence today, if not the fastest is the one written by Technical Systems Consultants (TSC) of Indiana, and the COMPUMAX uses it. It is located in 6 EPROMs, and is indeed superfast.

The TSC BASIC incorporates two main colorgraphics commands, the PLOT and the DRAW. The screen is divided into two axes, the X and Y axes, addressable, in low resolution, into 32 by 64 segments and, in high resolution, into 250 by 190 segments. The designation of the various colours is given by number. A typical example for designing a bar graph would be:

```
10 PLOT (clears the screen)
20 MODE,0 (low resolution graphics)
30 DRAW, 4, 0,0,0,30(this will give you a
perpendicula bar from top to bottom, red in
colour)
```

By setting various draws with the appropriate colours, you can easily come with a bar graph. The colorgraphics in the COMPUMAX are truly excellent.

THE DISK OPERATING SYSTEM

The COMPUMAX uses the TSC FLEX DOS which is fully equipped with disk BASIC, assembler, dissassembler, relocator, math package, etc. The disk controller circuit, standard on the main board, accesses the disk drives quite easily. All one has to do, while in the operating system monitor, hit key P, and he or she is on the DOS. Then, of course, you will have to wait a few seconds until FLEX DOS is loaded into memory.

One inter-sting feature of the computer is that, while you are under DOS, you can actually deactivate the memory space taken up by the PROM BASIC, and use that space for your needs.

However, the interesting software arsenal of the COMPUMAX does not stop here. It will be equipped with PASCAL, FORTRAN, and COBOL in the 3rd quarter of 1980.

A truly magnificent system.

sinclair
ZX80

NEW SINCLAIR COMPUTER FOR UNDER £100

EXTREMELY portable, measuring 9 x 7 x 2 inches max (218 x 170 x 50mm) and weighing 12 ounces (340g), the ZX80 is intended for use at home, work, college or school. As well as providing a personal insight into computer programming and applications for the business executive, and a vital initiation to computers for students and school children, the ZX80 is a powerful tool for the experienced user.

A 130 page instruction manual is provided to simplify learning by direct response. The manual includes a course in BASIC programming—the established standard high level language for personal computers—used by the ZX80.

To ensure maximum flexibility of use, the ZX80 has been built without a dedicated VDU (visual display unit), but with the facility to be plugged directly into the aerial socket of any domestic colour or black and white television. This also enables screens of a variety of sizes to be used.

A conventional home cassette player is used to store programs.

For business or industrial use, the Sinclair ZX80 can be coupled to any type of computer peripheral, such as a printer. It is envisaged that the unit will also be incorporated in a variety of industrial systems—as a machine tool control, for example.

A key advance is the design of a single super ROM containing the BASIC interpreter, character set, operating system and monitor. The ZX80's 1K byte RAM is equivalent to 4K bytes in a conventional computer.

Program entry is via a touch-sensitive, typewriter configuration, alpha-numeric keyboard which features single stroke key word entry eliminating much tiresome typing.

The Sinclair computer has powerful edit facilities and every statement line is syntax checked as it is entered at the bottom of the screen so that only syntactically correct lines can be added to the program list at the top. A marker identifies a syntax error and this unique feature will speed the production of an error-free program and is seen as of particular value for beginners.

The ZX80 display—black on white for clarity—consists of 24 lines of 32 characters each.

Featuring high resolution graphics with 24 standard graphic symbols available, the computer also allows any alpha-numeric or graphic symbol to be reversed.

The Sinclair ZX80 can be purchased in kit form at £77.95 from Science of Cambridge Ltd (a Sinclair company) and built versions will be availble during early March for £99.95 inc VAT. The prices include the manual but exclude mains adaptors at £8.95.

SINCLAIR PERSONAL COMPUTER TECHNICAL INFORMATION

1. The Sinclair personal computer employs the Z80A microprocessor chip supplied by the Nippon Electric Company (NEC).

2. The Sinclair BASIC interpreter provides some important advantages. Examples include:—

   —Single stroke key word entry. In most computers it is necessary to type out any key word in full, ie PRINT when print is required. Only that key with the word 'PRINT' above it need be hit with the ZX80, and no shift is required since the machine anticipates a key word knowing that it follows a line number. This feature eliminates much tiresome typing.

   —Powerful edit facilities.

   —Every statement line is syntax checked as it is entered and only syntactically correct lines can be added to the program list. The line being typed in appears at the bottom of the screen and only joins those lines at the top if it is free of syntax errors. If there is a syntax error then a marker identifies it, so that it may be eliminated. This unique feature helps ensure the production of an error-free program and is of particular value for beginners.

   —The BASIC has good string handling capability. There can be up to a maximum of 26 string variables and the strings can be of any length. All rational tests may be used on the strings so that, for example, strings may be compared. The machine has string input so that the computer can reqest a line of text (string) when necessary. Strings do not need to be dimensioned, which is an unusual benefit, and up to a maximum of twenty-six single dimension arrays are possible.
   Nesting for loops is also permitted up to a maximum of twenty-six. Integer variables may be of any length. The BASIC is capable of handling full Boolean arithmetic, conditional expressions, etc.

   — Built-in fuctions are:—
   CHR$
   STR $
   TL $
   PEEK
   CODE
   RND
   USR
   ABS

   —The randomise function is of particular value for games and secret codes etc., as well as being powerful in more serious applications.

   —A timer is available under program control.

   —PEEK and POKE enable the entry of machine code instructions and USR causes a jump to a users machine language sub-routine.

3. The computer has a complete alpha-numeric keyboard using the standard 'typewriter' configuration.

4. It has high resolution graphics with 24 standard graphic symbols being available. Any alpha-numeric or graphic symbol may be reversed. The display consists of 24 lines of 32 characters each.

5. An expansion bus available at the back by edge connectors brings out all lines including address and data lines so that there is no restriction on the extension of the machine. Plug in boards are available to increase the memory capacity. Memory expansion boards, which take up to 3K bytes, are £12.00 each and RAM memory chips of standard 1K bytes capacity are £16.00.

NEW APPOINTMENT FOR COMPUTER RETAILERS ASSOCIATION

THE Computer Retailers Association has appointed Mrs. Helen M. W. Gibbons as Assistant Secretary, reporting to the Secretary Mr. T. Moore of Newbear. This appointment is one of vital importance to the Association as it will provide it with a full equipped permanent Secretariat which will give it a sound and well organised base from which to grow.

Mrs. Gibbons, who has many years' experience as an Association Secretary, sees the newly formed Computer Retailers Association as one of the most dynamic in the industry.

At its formation a few months ago, twenty two companies formed the nucleus of its membership and now, with the appointment of a Secretariat the CRA, which is the only body representing the micro-computer industry is soon to embark on a full programme of activities which is likely to bring it to the attention of Computer Retailers all over the country and thereby attract a much larger membership.

Membership is open to companies in the computer field who have a significant interest in supplying microcomputers or related products and services to end users and who have a permanent display area where their products and services can be effectively demonstrated.

Mrs. Gibbons would be delighted to answer enquiries from potential members and she may be reached at Computer Retailers Association, Owles Hall, Buntingford, Hertfordshire, SG9 9PL—Telephone Royston (0763) 71209.

NASCOM SYSTEM 80

NASCOM Microcomputers launched a desk top microcomputer system called System 80 which combines many of the company's widely acknowledged products with a number of new boards and peripherals.

With the exception of the IMP (impact matrix printer) which is only supplied built, all the products can be supplied in built or kit form. The new floppy disc system will be available with one or two drives. The second drive can be easily added at a later date if the user's initial resources are limited.

CPU

The new system is based on the now well established Nascom 2 Microcomputer.

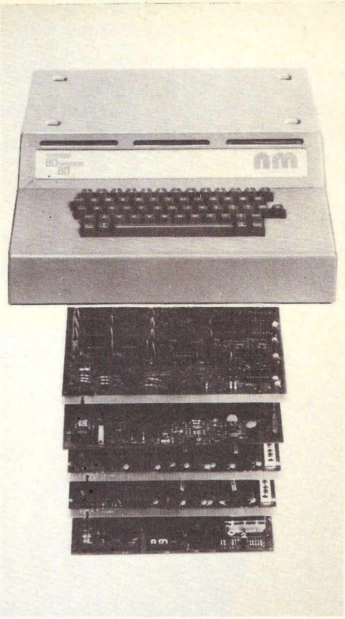
Housing

Within the System 80 housing is a frame racking that holds a NAS BUS motherboard, a power supply (3 amp or 5 amp depending on the choice of boards to be fitted), the CPU board, and up to four 8 in x 8 in expansion boards. Provision is made for external connection direct to the boards concerned. The Nascom 2 keyboard fits snugly in the cutout provided.

The housing is moulded from glass reinforced plastic, which combines lightness with high strength, and is available in a choice of colours. A TV or monitor can be placed on top of the housing—however this surface has been designed with recesses to accept the feet of an expansion housing which is being designed to hold five more boards. Using 78 way cable the two motherboards may be connected.

Floppy Discs

Another totally new product, the System 80 floppy disc system will be supplied with built and tested parts which can be bought individually or as a complete system. These include an assembled controller card (controls 4 drives), power supply unit, 5¼ in doubled sized, double density drives, enclosure and accessories for mounting two drives and the PSU. The industry standard CP/M disc operating system will also be available. Each drive provides 280K bytes of formatted storage.



NASCOM PROGRAMMABLE CHARACTER GENERATOR BOARD

THIS product is on an 8" x 8" P.C.B. which is NAS BUS compatible. It fits either a Nascom Frame or a 'Systems 80' microcomputer housing. There are 2K bytes of 2114 static RAM which is used as a programmable character generator.

The Nascom Graphics ROM may be relocated on this board. This gives the user the chance of software selection between the block graphics ROM and his own high resolution graphics at any time.

The high resolution graphics operate on a cell structure. Each Character cell is made up of

Nascom 1—128 dots
Nascom 2—112 dots

The user can produce 128 different cells to this dot level in the 2k RAM.

Each cell, once defined, may be displayed anywhere, and any number of times, on the screen at the same moment, up to a maximum screen capacity of 768 cells.

(48 x 16)

Dot resolution on the

Nascom 1 384 x 256 98304
Nascom 2 384 x 224 86016

A Z80 port is allocated to control the operational functions. These include RAM/ROM select, write protect etc.

Standard Alpha/Numeric characters and block and high resolution graphics may be intermixed on the screen. This board is designed to be compatible with the colour board.

NASCOM COLOUR BOARD

THIS board has various options.

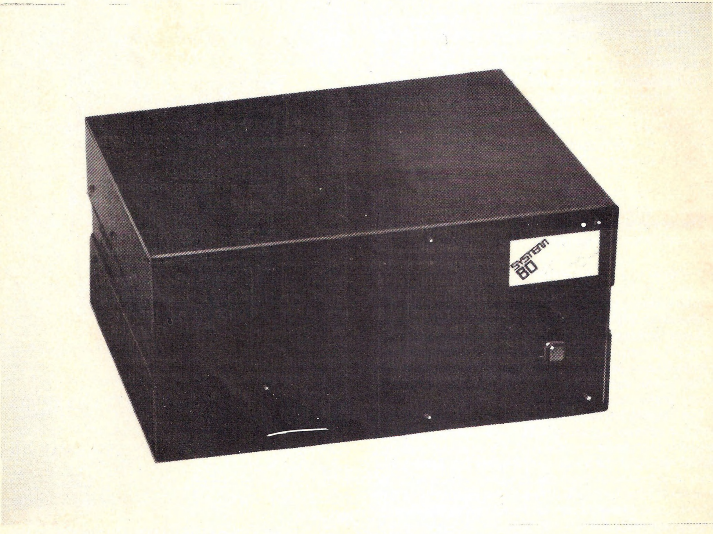
High or low resolution with PAL, SECAM, NTSC or RGB output.

High resolution uses 6K Static RAM (4118) and gives a choice of 16 colours. The foreground and background colours are definable on a 96 x 48 Matrix giving 4608 definable points.

In conjunction with the Programmable Character Generator board, and using the foreground and background capability, it is possible to use colour to an even higher apparent resolution.

Low resolution reduces the matrix to 48 x 48 and uses only 3K of Static RAM (MK 4118).

One of the Z80 ports is allocated to control the operational functions. These include colour on/off, write protect, high/low resolution.



NASCOM INPUT/OUTPUT BOARD

THIS product is one an 8" x 8" P.C.B. which is NAS BUS compatible. It fits either a Nascom frame or a 'System 80' microcomputer housing.

It is through plated and has all the necessary support chips.

When it is fully populated it will support.

3 x MK 3881 PIO
1 x MK 3882 Counter Timer
1 x 6402/UART

P10

The MK 3881 provides 2 x 8 bit wide bi-directional ports. Each has two handshake lines. It may be fully interrupt driven and implements the Z80 interrupt priority daisy chain. The PIO pack includes a 26 way board connector and cable.

Counter Timer

The MK 3882 contains four counter/timer channels. It may by fully interrupt driven and implements the Z80 interrupt priority daisy chain.

UART

The 6402 will provide serial communication through the RS 232C interface. The baud rate is selectable between 110 and 9600. The UART pack includes a 1.8432 Mhz crystal and an MC 14411 baud rate generator.

The PIO, CTC and UART may be located anywhere in the Z80 port address space. The required 1/O decode is fed back for either Nascom 1 or Nascom 2.

NASCOM DYNAMIC RAM CARD

THIS product is on an 8" x 8" P.C.B. which is NAS BUS compatible. It fits either Nascom frame or a 'Systems 80' microcomputer housing.

The user has a choice of three options: 16K bytes, 32K or 48K or 4116 dynamic RAM.

The 4116 is a 1 x 16K bits and will run at 4Mhz with a 'WAIT' state.

The board has full decoding, buffering and dynamic memory support for all packages.

There are three banks of 16K bytes. Each 16K bank is relocatable to 4K boundaries.

There is an up grade kit for this product with the following features.

Write protect–Each 16K block of memory may be protected from CPU write by using a toggle switch with LED indication.

Page Mode–It is possible to use up to four 48K RAM boards in a system. As the CPU will only access 64K at one time, the RAM board in use is software selectable with separate read and write enables.

NASCOM FLOPPY DISC CONTROLLER BOARD

THIS product is on an 8" x 8" P.C.B. which is NAS BUS compatible. It fits either a Nascom Frame or a 'System 80' microcomputer housing.

The board is capable of driving up to four Siemens double density, double sided 5¼" Mini FLOPPY drives.

It uses the industry standard 1791 controller chip.

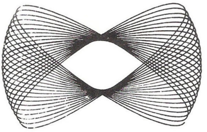
Data Transer does not rely on CPU interrupts wait states or D.M.A. The system uses real time loop transfer.

To maximise reliability a phase lock loop technique is used to synchronise data transfer from the drive to the 1791.

The board has simple test links which make the set up operation extremely simple.

The board has various link optional available. The user may decide on single or double sided discs and single or double density. He may also run his CPU at 2 Mhz or 4 Mhz.

1.	Systems 80 Box	85.00
	3 Amp PSU	30.00
	Nascom 2 Board	225.00
	32K RAM Board	165.00
		505.00
2.	System 80 Box	85.00
	3 amp PSU	30.00
	Nascom 2 Board	225.00
	48K RAM Board	225.00
	P.C.G.	90.00
	Colour Card (Highest Res)	165.00
		820.00
3.	System 80 Box	85.00
	5 amp PSU	40.00
	Nascom 2 Board	225.00
	48K RAM	225.00
	P.C.G.	90.00
	Complete Twin Disc Set	690.00
		1355.00
4.	System 80 Box	85.00
	5 amp PSU	40.00
	Nascom 2 Board	225.00
	3 x 48K RAM Boards	675.00
	Complete Twin Disc Set	690.00
		1715.00
5.	System 80 Box	85.00
	5 amp PSU	40.00
	Nascom 2 Board	225.00
	4 x 48K RAM Board	900.00
		1250.00



How to get your
LIVERPOOL SOFTWARE GAZETTE
regularly

Imagine the disastrous effect on your life style if you missed a single issue. The possibility of trauma is easily eliminated by the simple expedient of acquiring a regular subscription at the all time bargain price of £6.00 for the next twelve scintillating issues.

Don't miss the chance of a lifetime
Fill in the form below.

Commercial and educational organisations requiring a large number of copies can buy in bulk at advantageous rates from:

Computer Bookshop,
43-45 Temple Street,
Birmingham.

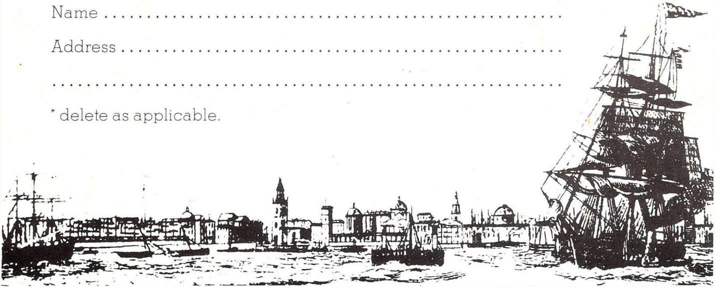
Please send me twelve issues of the Liverpool Software Gazette starting with the first/second/third/fourth issue. Cheques and PO's should be made payable to Liverpool Software Gazette and sent to us at 14 Castle Street, Liverpool.

Name ....

Address ....

....

* delete as applicable.



Printed by MERSEY MIRROR LTD., Media House, 34 Stafford Street, Liverpool L3 8LX, Tel: 051-207 7113